Proceedings

Fifth Annual Conference on

AI, Simulation, and Planning in High Autonomy Systems

Distributed Interactive Simulation Environments

December 7–9, 1994
Gainesville, Florida

Sponsored by
The University of Florida
The Advanced Research Projects Agency (ARPA)
The Army Research Office (ARO)

IEEE Computer Society Press

The Institute of Electrical and Electronics Engineers, Inc.

MASTER COPY KEEP THIS COPY FOR REPRODUCTION PURPOSES

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE October 1995	3. REPORT TYPE AND DATES COVERED Final 20 Sep 94 - 19 Sep 95

4. TITLE AND SUBTITLE

Distributed Interactive Simulation Environments

5. FUNDING NUMBERS

DAAH04-94-G-0414

6. AUTHOR(S)

Paul A. Fishwick

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

University of Florida
Gainesville, FL 32611

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

U.S. Army Research Office
P.O. Box 12211
Research Triangle Park, NC 27709-2211

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

ARO 33598.1-MA-CF

11. SUPPLEMENTARY NOTES

The views, opinions and/or findings contained in this report are those of the author(s) and should not be construed as an official Department of the Army position, policy, or decision, unless so designated by other documentation.

12a. DISTRIBUTION/AVAILABILITY STATEMENT

Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

This represents the Fifth AI, Simulation, and Planning Conference for high autonomy systems. High autonomy systems are large scale dynamic systems involving many interacting intelligent or controlled entities. Past conferences were held in Tucson (Arizona), Cocoa Beach (Florida); and Perth, Australia.

Large scale simulation models are increasingly executed within parallel and distributed computing environments. Distributed Interactive Simulation (DIS) directly involves the human in the simulation loop, and contains the real-time communication of heterogeneous simulators spread throughout wide geographical areas. Research in distributed simulation has taken place across many fronts: (1) Military DIS IEEE standard and workshops; (2) Continuous model parallelization; and (3) Discrete model (PDES) parallelization. The purpose of this conference is to focus on basic research problems in the overall area of distributed simulation with an emphasis on problems occurring in interactive environments.

14. SUBJECT TERMS

15. NUMBER OF PAGES

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT UL

Proceedings of the

Fifth Annual Conference on

AI, Simulation, and Planning in High Autonomy Systems

Distributed Interactive Simulation Environments

Proceedings of the

Fifth Annual Conference on AI, Simulation, and Planning in High Autonomy Systems

Distributed Interactive Simulation Environments

December 7–9 , 1994
Gainesville, Florida

Sponsored by

University of Florida
The Advanced Research Projects Agency (ARPA)
The Army Research Office (ARO)

IEEE Computer Society Press
Los Alamitos, California

Washington • Brussels • Tokyo

IEEE Computer Society Press
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264

The papers in this book comprise the proceedings of the meeting mentioned on the cover and title page. They reflect the authors' opinions and, in the interests of timely dissemination, are published as presented and without change. Their inclusion in this publication does not necessarily constitute endorsement by the editors, the IEEE Computer Society Press, or the Institute of Electrical and Electronics Engineers, Inc.

IEEE Computer Society Press Order Number 6440-02
Library of Congress Number 94-76639
ISBN 0-8186-6440-1 (paper)
ISBN 0-8186-6441-X (microfiche)

Additional copies may be ordered from:

IEEE Computer Society Press
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264
Tel: +1-714-821-8380
Fax: +1-714-821-4641
Email: cs.books@computer.org

IEEE Computer Society
13, Avenue de l'Aquilon
B-1200 Brussels
BELGIUM
Tel: +32-2-770-2198
Fax: +32-2-770-8505

IEEE Computer Society
Ooshima Building
2-19-1 Minami-Aoyama
Minato-ku, Tokyo 107
JAPAN
Tel: +81-3-3408-3118
Fax: +81-3-3408-3553

Editorial production by Penny Storms
Cover by Joseph Daigle - Schenk/Daigle Studios
Printed in the United States of America by KNI, Inc.

The Institute of Electrical and Electronics Engineers, Inc.

Table of Contents

TRACK 1

Session 1A: Hybrid Systems

Session 1B: Modeling and Dynamics

Session 1C: Terrain Modeling and Reasoning

Session 1D: Network Analysis

Session 1E: Intelligent Agents

Session 1F: Planning and Decision Making I

Session 1G: Planning and Decision Making II

TRACK 2

Session 2A: DEVS Formalism: Simulation Engines and Performance Modeling

Session 2B: DEVS Formalism: Modelling Methodology

Session 2C: DEVS Formalism: Manufacturing Applications

Session 2D: DEVS Formalism: Discrete Event Systems

Session 2E: DEVS Workshop Working Session

Session 2F: Applications I

Session 2G: Applications II

Session 2H: Test and Evaluation

Greetings

Welcome to the Fifth Annual Conference on AI, Simulation, and Planning (AIS 94) held at the University Centre Hotel adjacent to the University of Florida campus. Every year, the AIS conference adopts a different theme, which targets a central research problem in the area of computer simulation. This year's theme is "Distributed Interactive Simulation (DIS) Environments." Since simulation involves intelligent as well as non-intelligent objects, this conference includes technical themes reflecting a combined simulation/AI approach.

As simulation models are designed with greater numbers of components and sub-components, simulation researchers need to find ways to efficiently execute models. Moreover, many simulation models contain a "human in the loop," therefore interactivity plays a key role during simulation. There are several major technical hurdles in DIS including 1) how to design large-scale networked models; 2) how to make models operate in real time when training is the simulation goal; and 3) how to effectively partition the mathematical models and data sets to reduce network traffic and speed up the simulation. All papers in this proceedings address these problems, and more, under the umbrella of DIS.

I would like to acknowledge several individuals who have helped to create this conference. In terms of co-support, I would like to thank two organizations: the Advanced Research Projects Agency (ARPA) and the Army Research Office (ARO). Without their financial and technical assistance, this conference would not be possible. Dennis McBride (ARPA) and Jagdish Chandra (ARO) have made many valuable technical suggestions, and their guidance is greatly appreciated. Ole Nelson of the University of Florida Department for Continuing Education/Conferences has been a most valued collaborator for local conference management, and has offered friendly and timely assistance with all conference-related matters. Edna Straub and Penny Storms of the IEEE Computer Society have helped to make the hardcopy proceedings a reality, and I spent many hours with Perri Cline, also with the IEEE Computer Society, on the technical aspects of storing the proceedings of this conference on an Internet-accessible IEEE Computer Society node in hypermedia and Postscript. Perri and the IEEE Computer Society are leading the way for the next generation of online proceedings, where readers can browse full text/graphics conference articles online using World Wide Web (WWW) client programs.

I wish you a pleasant stay and am sure that you will walk away from the conference with some original questions answered, and new questions to ponder. Welcome to "Gator Country" — otherwise known as Gainesville, Florida!

Paul A. Fishwick
Conference Chair
Email: fishwick@cis.ufl.edu

Conference Description

This represents the Fifth AI, Simulation, and Planning Conference for high autonomy systems. High autonomy systems are large scale dynamic systems involving many interacting intelligent or controlled entities. Past conferences were held in Tucson (Arizona), Cocoa Beach (Florida); and Perth, Australia.

Large scale simulation models are increasingly executed within parallel and distributed computing environments. Distributed Interactive Simulation (DIS) directly involves the human in the simulation loop, and contains the real-time communication of heterogeneous simulators spread throughout wide geographical areas. Research in distributed simulation has taken place across many fronts: (1) Military DIS IEEE standard and workshops; (2) Continuous model parallelization; and (3) Discrete model (PDES) parallelization. The purpose of this conference is to focus on basic research problems in the overall area of distributed simulation with an emphasis on problems occurring in interactive environments.

Committees

Conference Chair

Paul A. Fishwick
University of Florida

Organizing Committee

Jerzy W. Rozenblit
University of Arizona

Bernard P. Zeigler
University of Arizona

Program Committee

François E. Cellier, *University of Arizona*
Sandra Cheung, *Institute for Simulation and Training*
Paul Davis, *Rand Corporation*
Tom DeFanti, *University of Illinois at Chicago*
Stephen Downes-Martin, *David Sarnoff Research Center*
Adel Elmaghraby, *University of Louisville*
Richard Fujimoto, *Georgia Institute of Technology*
Dorota Kieronska, *Curtin University of Technology, Australia*
Tag Gon Kim, *Korea Advanced Institute of Science and Technology*
Jason Lin, *Bellcore*
Sven Erik Mattsson, *University of Lund, Sweden*
Duncan Miller, *Massachusetts Institute of Technology*
Michael Moshell, *University of Central Florida*
David Nicol, *College of William and Mary*
Tuncer Oren, *University of Ottawa, Canada*
Mikel Petty, *Institute for Simulation and Training*
Herbert Praehofer, *University of Linz, Austria*
Ashvin Radiya, *University of Wichita*
Roger Smith, *Mystech Associates, Inc.*
Scott Smith, *Institute for Simulation and Training*
Svetha Venkatesh, *Curtin University of Technology, Australia*
Ben Wise, *SAIC Corporation*
David Wood, *MITRE Corporation*
David Zeltzer, *Massachusetts Institute of Technology*
Michael Zyda, *Naval Postgraduate School*

Reviewers

François E. Cellier
Sandra Cheung
Paul Davis
Tom DeFanti
Stephen Downes-Martin
Adel Elmaghraby
Paul A. Fishwick
Richard Fujimoto
Dorota Kieronska
Tag Gon Kim
Jason Lin
Sven Erik Mattsson
Duncan Miller
Michael Moshell
David Nicol
Tuncer Oren
Mikel Petty
Herbert Praehofer
Ashvin Radiya
Jerzy W. Rozenblit
Roger Smith
Scott Smith
Svetha Venkatesh
Ben Wise
David Wood
David Zeltzer
Bernard P. Zeigler
Michael Zyda

Session 1A:

Hybrid Systems

Hybrid Systems and Distributed Interactive Simulations

John James
Intermetrics, Inc.
7918 Jones Branch Dr.
Suite 710
McLean, VA 22102 USA

james@potomac.wash.inmet.com

Anil Nerode
Mathematical Sciences Institute
Whitehall
Cornell University
Ithaca, NY 14853-7901 USA

nerode@mssun7.msi.cornell.edu

Wolf Kohn
Intermetrics, Inc.
1750 112th Ave NE
Suite D-151
Bellevue WA 98004 USA

wfk@minnie.bell.inmet.com

Jagdish Chandra
Director, Mathematical and Computer Sciences Division
Army Research Office
P. O. Box 12211
Research Triangle Park, NC 27709-2211

chandra@aro-emh1.army.mil

Abstract: The Army has set a goal of enhancing battlefield effectiveness by fielding a digital division by 1998 and has started a sequence of field exercises to investigate how new applications of digital technology will affect military operations. These transition activities are part of preparing the Army to fight third-wave warfare - information-age warfare. The result will be a 21st-century Army, digitized and redesigned to fight the wars of the next century. *Distributed Interactive Simulation* (DIS) is seen as a key technology in determining and analyzing alternatives for digitizing the battlefield. DIS is a rapidly changing field. For over twenty years the Army has been using computer-assisted tactical engagement simulations and distributed interactive simulations to enhance training and evaluate engagement alternatives. Based on this experience, an emerging vision is the application of advanced information systems technology to create a shared situational awareness (visualization) of the battlefield. Realization of shared awareness will support faster-paced operations through real-time force synchronization. The Louisiana Maneuvers (LAM) initiative will use the joint AMC and TRADOC Battle Labs to investigate some of the alternatives. Given the rapid changes in computer capabilities, communications bandwidth, and software complexity, it has been unclear what the mid- and far-term technical opportunities and challenges are in applying results from the ongoing information systems revolution to improve battlefield effectiveness. Substantial improvements in current DIS technologies are needed to enable professionals at widely distributed sites to interact simultaneously through simulators, simulations, and deployed systems in a common joint synthetic operational environment. In this article we discuss how one of the foundational technologies supported by the U. S. Army Research Office, hybrid systems technology, can support closing some of the DIS technological gaps and thus help to analyze alternatives for realization of Force XXI..

1. Introduction: Army Battle Labs, the National Simulation Center, and the Louisiana Maneuvers (LAM) office are principle sources of Army requirements for DIS. The vision for the future of Advanced Distributed Simulation (ADS) includes creation of synthetic theaters of operation shared and simultaneously operated on by the Services, CINCs, Joint Task Forces, Joint Staff, and Defense community. Realization of the vision of a synthetic theater of war (STOW) fully depends on the creation of interoperable simulators, simulations, and fielded systems that *realistically* represent warfighting concepts, doctrine, forces and weapon systems of friendly, neutral and opposing forces [2]. Distributed Interactive Simulations (DIS) have provided a wide variety of realistic training and analysis applications, but cannot currently be relied upon to create the STOW because the architecture is known to have problems with *synchronization, interoperability* and *scalability* [3,4,5]. General Gordon R. Sullivan, Chief of Staff, U. S. Army, has recently asserted that

0-8186-6440-1/94 $04.00 © 1994 IEEE

"Force XXI will represent a new way of thinking for a new wave of warfare.[6]." Overcoming DIS technology shortfalls is a necessary step in being able to analyze alternatives in creation of new formations that operate at greater performance levels in speed, space, and time.

1.1. Shortfall in synchronization: Advances in technology are needed to improve simulation realism and accuracy by correcting shortfalls in synchronization of live virtual and constructive simulations. Currently, information generated by one simulation is not reliably shared with other simulations in time to achieve realistic interaction (such as a tank being simulated in one location driving smoothly over a crater created by another simulation, or a tactical radio which depends on line-of-sight "communicating" with another radio that is over 200 miles away). Realistic synchronization of simulations is necessary to engender the level of confidence in simulation results needed to provide strategic direction for the Army.

The Army Master Plan for DIS asserts that DIS will play a key role in strategic direction for the Army by enabling evaluation and analysis of strategic concepts, military options and mission needs. The Louisiana Maneuvers (LAM) process provides a capability for the Army's senior leadership to guide, formulate and assess military options for continuously improving Army capabilities at the strategic and operational levels of war. To meet LAM mission needs, future simulations must represent the complimentary capabilities of all Services in all missions ranging from a full-scale theater operation to a small-scale peacekeeping mission [2]. Given General Sullivan's intent to build formations that operate at greater performance levels in speed, space, and time and the shortfall of the existing DIS architecture to accurately depict formations maneuvering at current rates, synchronization of simulations which will enable analysis of innovative alternatives of future formations is a priority research issue.

1.2. Shortfall in interoperability: Advances in technology are needed to improve interoperability of existing diverse simulations. This will increase realism and decrease costs of constructing large-scale simulations like WarBreaker [3]. Many large-scale simulations similar to Warbreaker will be needed to support analysis of alternatives for horizontal technology integration envisioned for STOW. Excellent efforts are underway to improve interoperability by implementing standard processes for Protocol Data Unit (PDU) interfaces and use of the Aggregate Level Simulation Protocol (ALSP). However, these efforts to improve the integration process have been hampered by the fact that there has not been a mathematical framework for simultaneously analyzing safety, security, and other logical requirements, while considering temporal and spatial constraints. Ongoing ARO research efforts provide foundational technology for solving basic problems in the correlation of time and space in the synthetic environment, thereby achieving synchronization and interoperability.

Synchronization and interoperability are both key to Horizontal Technology Integration (HTI). HTI is the concept for designing major platforms that permit rapid replacement of common components and subsystems [2]. HTI requires greater attention early on to requirements trade-offs, baseline design trade-offs, and integrated development across platforms and subsystems. The existing DIS architecture does not support the level of interoperability needed for complex and critical trade-offs requiring close coordination and extensive interaction among combat development, materiel development, and materiel acquisition.

1.3. Shortfall in scalability: Advances in technology are needed to support scalability of simulations. Two or three orders of magnitude increase in the scale of the number of objects being simulated is estimated as being necessary to support the kinds of simulations envisioned for DIS [3]. Synthetic battlefields must represent the full dimension of ground, air, maritime and space operations across the entire spectrum of conflict and operations other than war [2]. The synthetic battlefields must expand representations of forces, units, systems, installations, logistics networks, terrain, environment, cultural features and people.

Maintaining consistent views of the battlefield at multiple locations and at multiple levels of abstraction is a major challenge. High levels of resolution are required for analysis of environmental effects on component and systems performance while low levels of resolution are needed at strategic and operational levels. Creating, updating, and interacting with the synthetic environment at both high- and low-levels of resolution simultaneously places extreemly difficult constraints on maintaining consistent interactions among simulations and models based on widely varying spatial and temporal scales.

1.4. Addressing technical shortfalls and achieving Verification, Validation and Accreditation The revolution in military affairs implied by the DIS vision depends on commanders trusting the simulation results; verification, validation and accreditation (VV&A) of new models must be rapid enough to meet the needs of national decision makers. DIS may require operation of a nested set of models or the results of the high-resolution models may be used as input to higher-order models. The realistic linkage of models and simulations, the real-time interaction between models of different levels of detail, and the need to maintain verification and validation for confederation of linked models are essential for DIS [2].

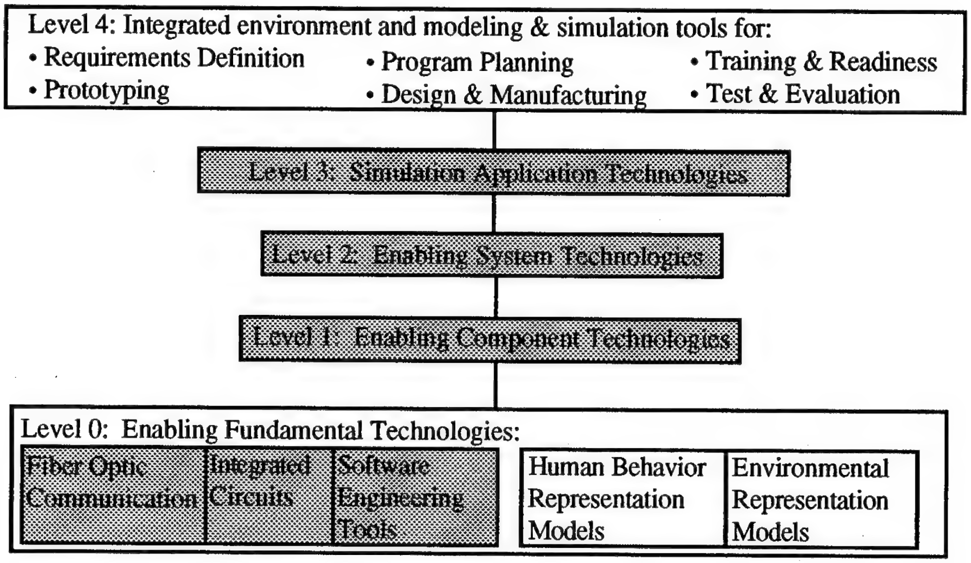

Level 4: Integrated environment and modeling & simulation tools for:
• Requirements Definition
• Prototyping
• Program Planning
• Design & Manufacturing
• Training & Readiness
• Test & Evaluation
Level 3: Simulation Application Technologies
Level 2: Enabling System Technologies
Level 1: Enabling Component Technologies
Level 0: Enabling Fundamental Technologies:
Fiber Optic Communication
Integrated Circuits
Software Engineering Tools
Human Behavior Representation Models
Environmental Representation Models

Figure 1. Simulation Enabling Technologies

During a recent ARO workshop academic, government and commercial representatives were informed concerning DIS requirements and technical shortfalls, and contributed to a technical discussion which centered around the potential of *hybrid systems* to correct some of the shortfalls. Hybrid systems are represented by models which are compositions of logic models (digital, linguistic, algebraic or finite-state-machine models) and evolution models (analog or differential operator models). A DIS is a hybrid system. Recent research results in mathematics and control theory suggest that hybrid systems may offer a path to high-safety, high assurance systems composed of trusted modules. The formal nature of hybrid systems theory supports dramatically lowering the cost of integrating heterogeneous simulations by greatly reducing the effort required to perform verification, validation and accreditation of incremental changes to trusted DIS components.

A technical shortfall discussed at some length at the workshop is a barrier to cost-effective implementation of ADS. The shortfall is *lack of an extractive methodology and an architecture for flexible interoperability of distributed, real-time information systems*. By extractive methodology we mean an extraction algorithm, tools and process for integration of existing simulation system components which will overcome a primary shortfall of the current technology which relies on experimentation to integrate heterogeneous systems. The new technology discussed to overcome this barrier is hybrid systems theory. Hybrid systems are those systems most appropriately described by an amalgamation of logical representations and evolution representations. A network of distributed interactive simulation (DIS) processes is a hybrid system. A need was identified to expand the hybrid systems results achieved thus far and investigate its application to achieve the interoperability of existing and future DIS. It was not deemed appropriate to apply hybrid systems theory to existing simulations since rewriting these systems would be an enormous task. Instead, it was thought appropriate to apply the results of hybrid systems theory to a crucial problem in cost-effective enhancement of the existing infrastructure: *reactive, scalable interoperability of current and future simulations.*

2. Hybrid systems and integration of heterogeneous models: A recent Defense Science Board Summer Task Force studied the impact of advanced distributed simulation technology on service and joint readiness. The report of the board clearly indicated a *split* in the enabling fundamental technologies for software development (Level 0 of Figure 1). The split is the explicit dependence upon two very different kind of models and simulations: *logical models* (human behavior representation models) and *evolution models* (environmental representation models). Engineers, computer scientists, and mathematicians deliberately trade off between the two different kinds of models for different system components (or for different levels of aggregation of the same component) as part of the system development process. System integration often centers around ensuring that the different kinds of models are compatible for different environmental operating conditions (modes of operation).

A foundational issue for improving DIS capabilities is creation of a technology for *constructive integration of diverse models.* The integration of human behavior representation models and environmental representation models is currently achieved through expensive experimentation. Furthermore, this experimentation *limits the scalability* of simulation architectures since without some kind of constructive approach, incremental expansion of any architecture through addition of new components must be *experimentally verified.* What is needed is a methodology and an architecture for reliably ***constructing*** new versions of an existing architecture as new components are incrementally added to the existing architecture. Such a methodology will be the key to realization of the requirement for building reconfigurable simulations in support of "train the way you will fight." It will also be the key to maintaining consistent data as diverse models are mixed and matched in the reconfiguration process and also to the verification and validation of the confederation of models.

Thus, we believe that one of the most challenging problems facing DIS, the solution to which is most likely to lower costs, is the verification of software systems that use both logical (e.g. human behavior representation) and evolution (e.g. environmental representation) algorithms. These models use fundamentally different mathematical tools. Cognitive models of human behavior are built using linguistic tools which depend on the set-based mathematics of algebraic topology. Models of the physical environment are built using simulation tools which support experiments with compositions of set-based, linguistic (logical) models and continuum-based models which depend on the mathematics of differential operators. Experiments are necessary to determine the behavior of the composition of models for safety, reliability and performance constraints. The general approach currently used for verification is to explore design failure modes and track correction of bugs in the software until a comfort level is reached and success is declared. There is a nagging expectation that not all of the states of the computer finite-state machine have been visited and tested. Also, when new capabilities are added, new failure modes are created so the system must again be tested.

3. Overview of the Hybrid Systems Approach: The hybrid systems architecture development process expects to use software engineering processes and environments being developed under the ARPA DSSA program [7,8]. The DSSA approach emphasizes the role of the domain architect using CASE tools in the domain development environment to produce a reference architecture and reusable software components and the role of the application engineer using CASE tools in the domain-specific application development environment to apply the reference architecture and reusable components to produce an architecture instantiation. One hybrid systems architecture being considered consists of a collection of agents of two types : *Simulation Agents* and *Demand Agents* interconnected via a general purpose communications network (see Figure 1) [9]. This approach to control of hybrid systems, developed by Wolf Kohn, addresses the software design issue by building mathematical foundations (developed with Anil Nerode) and creating a tool for implementing a ***constructive algorithm for building automata which simultaneously comply with logical and evolution constraints.*** Experimentation to determine system equilibria is still required, as is the need to experimentally verify that the high-level logic meets the needs of the users. However, the need for exhaustive experimentation to analyze the result of combining high-level logic and low-level evolution representations is not required. Thus to the extent that high-level, logic models are "trusted" and low-level continuous-time models are "trusted", we can construct automata which are consistent with constraints from both kinds of models. Furthermore, if the logic or evolution models are not completely compatible with the system they are modeling, the procedure provides for formal mechanisms for tuning the structure of the logic and evolution models.

While space limitations preclude providing the scientific basis for hybrid systems claims. We provide the following summary of results for the Kohn-Nerode approach to multiple-agent hybrid control (see Figure 2):

- The formulation gives a precise statement of the DIS communication network control problem in terms of multiple agent hybrid declarative control. The approach characterizes the problem via a knowledge base of equational rules that describes the dynamics, constraints and requirements of the simulations being controlled (channels, switching modes, customer characteristics, scheduling and planning strategies, etc.). This result holds promise for addressing the current shortfall of the Aggregate Level Simulation Protocol in reconciling differences among its component constructive simulations. It specifically provides the ability to accommodate the time evolution of simulation elements from those available today through the time of availability of the technologies planned and under development since the architecture emphasizes incremental construction.

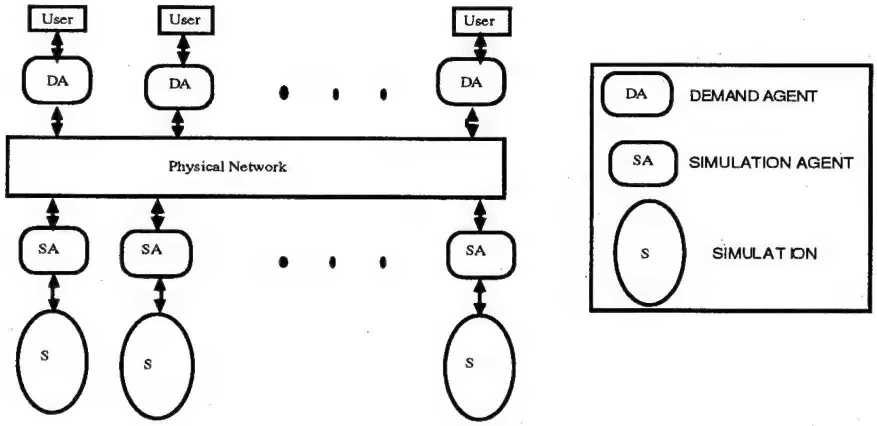


Figure 2. Multiple-Agent Hybrid Control Architecture for Advanced Distributed Simulation

- A canonical representation of interacting networks of controllers has been developed. Given a connectivity graph with N nodes (controllers) and the corresponding agent's knowledge bases, a network of 2N agents can be constructed with the same input-output characteristics, so that each agent interacts only with another (equivalent) companion agent, whose knowledge base is an abstraction of the knowledge in the network. Thus, in general, the multiple-agent controller for any network configuration is reduced to a set of agent pairs. This result supports the synchronization of the simulations to provide consistent data and the achievement of scalability.

- One agent of the agent pair maintains coordination with other agent pairs across the network. The agent of the pair which represents network information is called the Thevenin Agent, after the author of a similar theorem in electrical network theory. The proof carried out by the Thevenin Agent generates, as a side effect, coordination rules that define what and how often to communicate with other agents. These rules also define what the controller needs from the agent network to maintain intelligent control of its physical plant.

- The hybrid systems approach develops a canonical way to prove the theorem characterizing the desired behavior for each agent by constructing and executing on-line a finite state machine called the "proof automaton." This result is the basis for *constructing* simulations from existing component simulations and for the belief that the resulting architecture will support incremental expansion of new components with greatly reduced requirements for expensive experimentation to validate the new architecture. It is not expected that the need for experimentation will be entirely eliminated since the degree of "trust" in the newly composed architecture will depend on the rules for composition of the components. However, to the degree that the composition rules are correct, the methodology will be a *formally correct* composition of the components. Thus, the focus of the verification and validation effort will be raised to the component level and the results can be reused across the confederation of components.

In the full paper we will provide the hybrid systems model for Figure 2.

[1] "Force XXI - Battle Command" , Army, p. 23, May, 1994.

[2] Department of the Army, "Distributed Interactive Simulation Master Plan" (Draft), 1994.

[3] "Report of the Defense Science Board Task Force on Simulation, Readiness and Prototyping", Dr. Joseph V. Braddock and General Maxwell R. Thurman, Co-Chairmen, 21 Dec. 1992.

[4] "Proceedings of the U.S. Army Research Office Workshop on Hybrid Systems and Distributed Interactive Simulations", Research Triangle Park, NC, 28 Feb. - 1 Mar. 1994.

[5] DIS Steering Committee, "The DIS Vision - A Map to the Future of Distributed Simulation" (comment draft), (Margaret Loper, UCF, Chair; Steve Seidensticker, SAIC, DIS Vision Document Coordinator), Oct. 1993.

[6] "Force XXI - A Talk With the Chief", Army, p. 28-34, May, 1994.

[7] "DSSA Guidelines Draft 0.1", Proceedings of the DSSA Workshop IX, Teknowledge Federal Systems, MD, 28-29 March, 1994.

[8] Kohn, W., J. James and Anil Nerode, "Multiple-Agent Hybrid Control Architecture for the Target Engagement Process", Intermetrics Technical Report for the ARPA Domain-Specific Software Architectures (DSSA) Program, McLean, VA, March, 1994.

[9] Kohn, W., J. James, and A. Nerode, "Multiple-Agent Reactive Control of Distributed Interactive Simulations (DIS) Through a Heterogeneous Network", Proceedings of the U.S. Army Research Office Workshop on Hybrid Systems and Distributed Interactive Simulations, p. 100-140, Research Triangle Park, NC, 28 Feb. - 1 Mar. 1994.

[10] Butler, B., "DIS Architecture for Interoperability, Special Problem: Interoperability between Heterogeneous Visual Systems", Proceedings of the U.S. Army Research Office Workshop on Hybrid Systems and Distributed Interactive Simulations, p. 253-275, Research Triangle Park, NC, 28 Feb. - 1 Mar. 1994.

[11] Nerode A., Kohn W., " Multiple Agent Autonomous Control: A Hybrid Systems Architecture" Logical Methods In Honor of Anil Nerode's Sixtieth Birthday, N. C. Crossley , J. B. Remmel, M. E. Sweedler, Eds., Birkhauser, Boston, 1993.

Hybrid Systems: Models, Simulation, and Testing

Norman Coleman, Steve Banks
Robotics and Automation Laboratory
US Army Armaments Research, Dvelopment and Engineering Center
Building 95
Picatinny Arsenal, NJ 07806-5000

John James
Intermetrics, Inc.
7918 Jones Branch Dr.
Suite 710
McLean, VA 22102 USA

james@potomac.wash.inmet.com

Anil Nerode
Mathematical Sciences Institute
Whitehall
Cornell University
Ithaca, NY 14853-7901 USA

nerode@mssun7.msi.cornell.edu

Wolf Kohn
Intermetrics, Inc.
1750 112th Ave NE
Suite D-151
Bellevue WA 98004 USA

wfk@minnie.bell.inmet.com

Abstract: Realization of the vision of a synthetic theater of war (STOW) fully depends on the creation of interoperable simulators, simulations, and fielded systems that *realistically* represent warfighting concepts, doctrine, forces and weapon systems of friendly, neutral and opposing forces. Distributed Interactive Simulations (DIS) have provided a wide variety of realistic training and analysis applications but cannot currently be relied upon to create the STOW because the architecture is known to have problems with synchronization, interoperability and scalability.

The revolution in military affairs implied by the DIS vision depends on commanders trusting the simulation results. Additionally, the verification, validation and accreditation (VV&A) of new models must be rapid enough to meet the needs of national decision makers. We provide preliminary ideas concerning how a detailed model being develop as part of the Automation and Robotics program of the U. S. Army Armaments Research, Development and Engineering Center can be applied to address current problems in DIS by providing a timely approach to both simulation implementation and VV&A. Such an approach can be used to provide standard design guidance for linking models into seamless simulation exercises. Existing models can be incorporated by creating the appropriate simulation agents, thereby providing backward compatibility.

We will provide an overview of the architecture being developed and details concerning how such low-level, detailed models can be integrated into distributed, agent-based architectures to support early development of flexible test products. Such test products can be made to mature with the engineering design to provide a more realistic set of test products for VV&A of new systems.

1. Basis for Incremental Verification Validation and Accreditation: One approach to a scalable

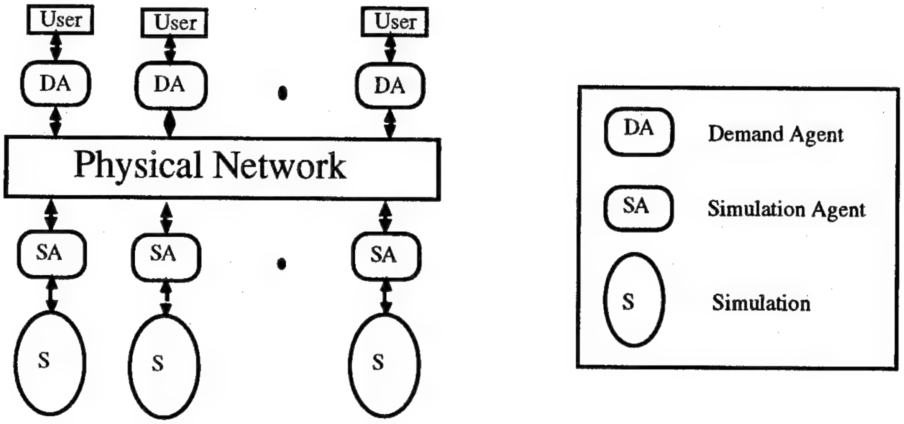


Figure 1. Multiple-Agent Hybrid Control Architecture

0-8186-6440-1/94 $04.00 © 1994 IEEE

simulation architecture consists of a collection of *Simulation Agents* and *Demand Agents* interconnected via a general purpose communications network (see Figure 1). Our approach to control of hybrid systems addresses the issue of maintaining a consistent instantiation of the architecture by building mathematical foundations and creating a tool for implementing a ***constructive algorithm for building automata which simultaneously comply with logical and evolution constraints.*** Experimentation to determine system equilibria is still required, as is the need to experimentally verify that the high-level logic meets the needs of the users. However, the need for exhaustive experimentation to analyze the result of combining high-level logic and low-level evolution representations is not required since we generate programs which are consistent with the logical and evolution constraints. Thus to the extent that high-level, logic models are "trusted" and low-level continuous-time models are "trusted", we can construct automata which are consistent with constraints from both kinds of models. Furthermore, if the logic or evolution models are not completely compatible with the system they are modeling, the procedure provides for formal mechanisms for tuning the structure of the logic and evolution models. The proof capabilities for each agent in the multiple-agent architecture allows its to determine whether or not the local behavior of the system (as viewed by the agent) satisfies the requirements (the continuity condition) and if not, to modify reactively its plan so that requirement satisfaction (agreement set) is achieved. However, the on-line capabilities do not provide an effective proof for determining if for the given goal class, the reachability set of the carrier manifold trajectories is abundant. By abundant we mean that, at each decision point, the agreement set is populated with at least one solution for all the command actions. This reachability problem is essentially the problem of the validation of the knowledge base of the agent.

2. Problem Domain: The Distributed Nature of Engagement of Multiple Targets by Multiple Weapon Systems

The Army intends to field a digital division by 1998. Capabilities of the digital division are yet to be fully determined. However, it is expected that future Army combat operations will increasingly involve coalition forces and that future Army missions will increasingly require conduct of operations other than war (OOTW), such as peacemaking, peacekeeping, humanitarian support, and humanitarian relief. Current plans to reduce force structure is driving the Army to investigate increasing flexibility of existing units to support a wide range of missions.

Innovative use of barriers to decrease mobility of opposing forces and active use of barriers to cause opposing forces to move in desired directions has been a historic discriminator between success and failure in war. Barriers, such as intelligent mines are often used to channel opposing forces into an area where they can be engaged by direct and indirect fire weapons of the combined arms team. Successful demonstration of command and control of advanced mines will provide commanders with a flexible, lightweight means of increasing combat effectiveness. The Army experts in mine warfare determine the critical operational issues and criteria for success of the intelligent minefield. Future efforts to coordinate results of our current demonstration of target engagement with the intelligent minefield would provide an opportunity to test concepts for dominating the maneuver battle with concepts for operational use of intelligent mines.

2.1. Architecture Development: Battlefield Environment Model

The *battlefield environment* (see Figure 2) consists of a Universe (a prespecified region of the plane, i.e. A closed surface of $R^{2)}$, two types of objects (Friendly objects and Foe objects), and the rules which determine the reaction (the evolution of the state over time) of friend or foe objects given the current state (of friend and foe objects) and the context of the operational situation (set of logical inputs). The set of possible battlefield scenarios is the set of **sequences of friend and foe actions** (from initiation of the battlefield simulation until the friend and foe objects are in a terminal state) and **associated contexts** . For the purposes of this demonstration, we will have a limited set of friend objects (always three objects) and foe objects and a limited number of rules (equational clauses - see Section 5.1.1 and [44]) which determine the evolution of the state of the system. The model is deliberately constructed in order to reflect some of the actual conditions which occur on the battlefield but only at a level necessary to demonstrate the reasoning and behavioral capabilities of multiple-agent hybrid control.

3. Conclusion The structure of the multiple-agent demonstration and the scope of multiple-agent hybrid control theory admit the construction of a high-fidelity model which will *not be achieved* in the current multiple-agent demonstration. However, we are currently implementing the steps necessary to achieve integration of a detailed, high-fidelity model of direct and indirect fire weapons with a distributed, low-fidelity, multiple-agent models of multiple weapons. A goal of the research is to explicitly investigate how such a formalism will support development of test products to assist in more flexible VV&A of complex simulation models.

[1] Kohn, W., J. James, and A. Nerode, "Multiple-Agent Reactive Control of Distributed Interactive Simulations (DIS) Through a Heterogeneous Network", Proceedings of the U.S. Army Research Office Workshop on Hybrid Systems and Distributed Interactive Simulations, p. 100-140, Research Triangle Park, NC, 28 Feb. - 1 Mar. 1994.

[2] Kohn, W., J. James and Anil Nerode, "Multiple-Agent Hybrid Control Architecture for the Target Engagement Process", Intermetrics Technical Report for the ARPA Domain-Specific Software Architectures (DSSA) Program, McLean, VA, March, 1994.

[3] Butler, B., "DIS Architecture for Interoperability, Special Problem: Interoperability between Heterogeneous Visual Systems", Proceedings of the U.S. Army Research Office Workshop on Hybrid Systems and Distributed Interactive Simulations, p. 253-275, Research Triangle Park, NC, 28 Feb. - 1 Mar. 1994.

[4] Kohn, W., J. James, R. Modes, and A. Nerode, "Multiple-Agent Reactive Control of Distributed Interactive Processes (DIS): An Overview",

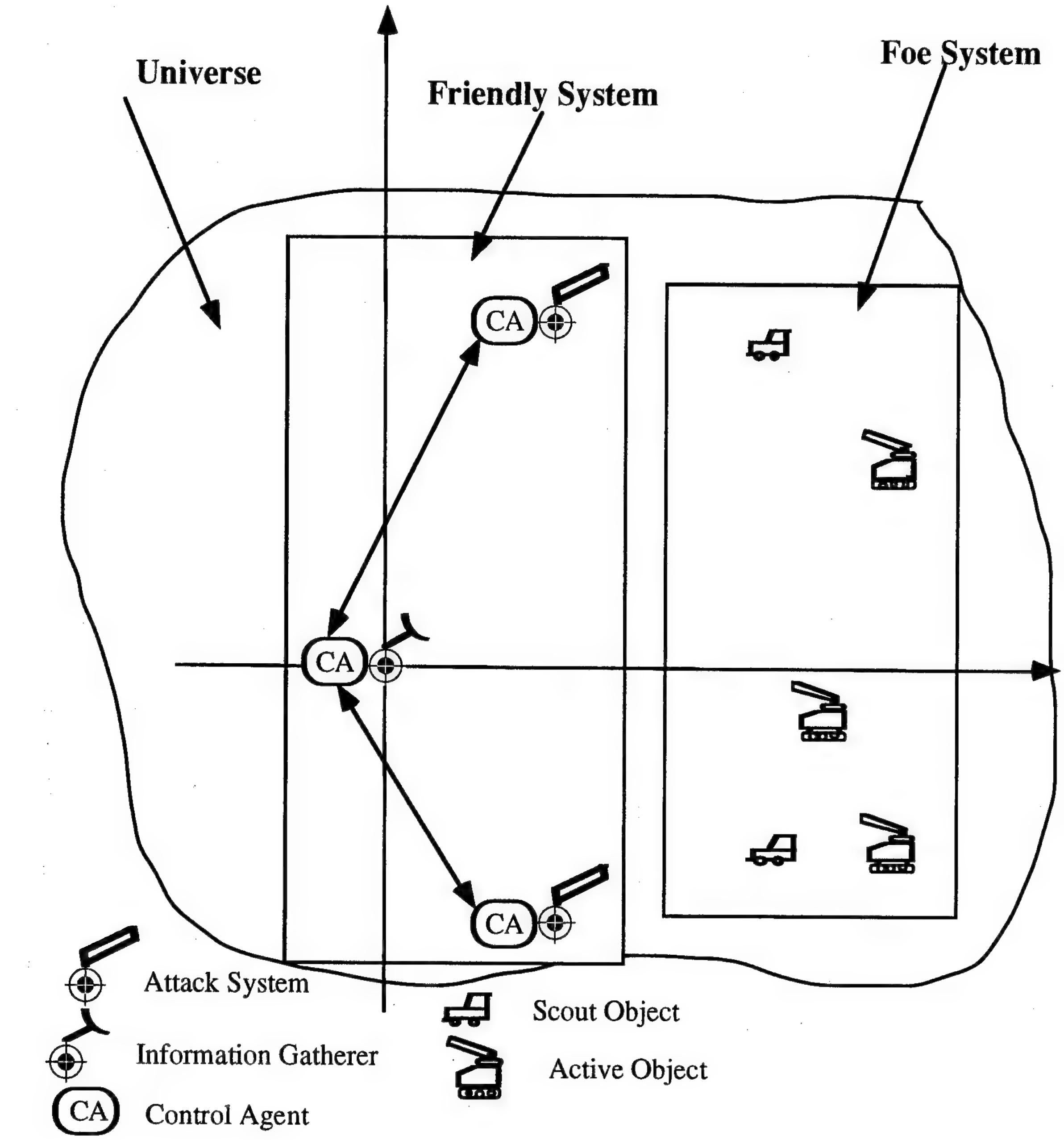


Figure 2. Battlefield Environment

Intermetrics, Inc., May, 1994.

[5] "Force XXI - A Talk With the Chief", Army, p. 28-34, May, 1994.

[6] Institute of Land Warfare, "Army Tests Info War", AUSA News, p. 7, June, 1994.

[7] Nerode A., Kohn W., " Multiple Agent Autonomous Control: A Hybrid Systems Architecture" Logical Methods In Honor of Anil Nerode's Sixtieth Birthday, N. C. Crossley , J. B. Remmel, M. E. Sweedler, Eds., Birkhauser, Boston, 1993.

Distributed Intelligent Control Theory of Hybrid Systems

Xiaolin Ge, Anil Nerode, Wolf Kohn, John James

Abstract: We discuss recent advances in multiple-agent, distributed control of interactive processes. Our multiple agent hybrid control architecture approach is a new technology with broad applicability since it integrates the application of logic (set-based) and continuous-time models of complex system behavior. The Kohn-Nerode approach for integration of hybrid systems emphasizes *on-line synthesis* of automata which will meet *current* constraints. A key feature of the approach is that the evolution of the behavior trajectory of the automata (e.g. the behavior trajectory of agent i) is continuous in the carrier manifold (explained below). We provide an example of our results by discussing the case of multiple agents involved in the engagement of multiple targets, the agent behavior is continuous with respect to the multiple engagement process models whose potentials determine the portion of time the process is active in a time interval.

1. A New Concept of System State

Hybrid System State: For purposes of our exposition, there are three cases which need to be considered in the characterization of unified information models represented in the computer. These cases collectively contain the kinds of information that describe the *state of the system*. The cases are: (1) information derived from monitoring continuous variables, (2) information derived from monitoring variables which normally evolve continuously but which may exhibit logical changes (jumps), and (3) information which is derived from logical variables (See Figure 1).

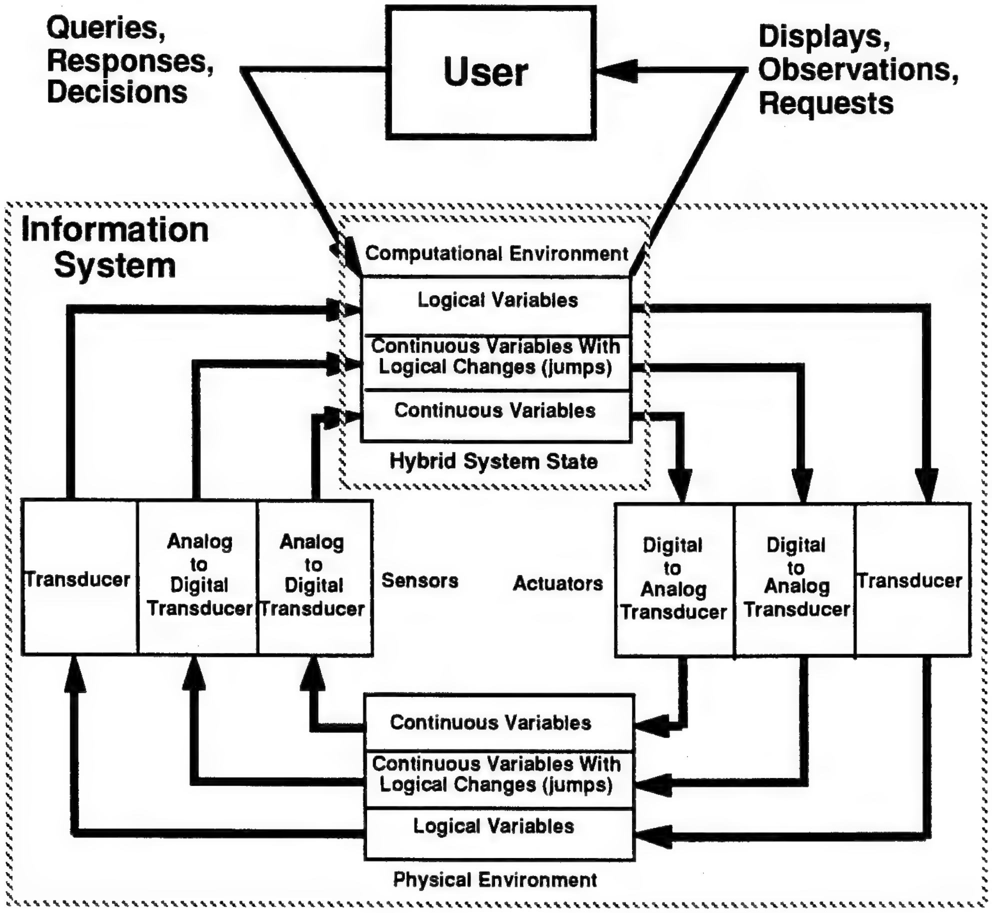

Queries,
Responses,
Decisions
User
Displays,
Observations,
Requests
Information System
Computational Environment
Logical Variables
Continuous Variables With Logical Changes (jumps)
Continuous Variables
Hybrid System State
Transducer
Analog to Digital Transducer
Analog to Digital Transducer
Sensors
Actuators
Digital to Analog Transducer
Digital to Analog Transducer
Transducer
Continuous Variables
Continuous Variables With Logical Changes (jumps)
Logical Variables
Physical Environment

Figure 1. Hybrid System State Representation

0-8186-6440-1/94 $04.00 © 1994 IEEE

To represent the state of the system in a computable model, the values of continuous variables must be approximated, as must the values of variables which are normally continuous but may occasionally exhibit jumps. This is achieved by A-D and D-A transducers . Exact representation of logical variables can be achieved using transducers. The data structure of the hybrid system state in the computational environment is a composition of logical and evolution variables. The state of the system in the physical environment is approximated by the state of the system in the computational environment (the hybrid system state). The information system of the user includes direct observation of the physical system as well as information available from the user interface of the computational environment. The hybrid system state evolves over time as the physical environment is altered by the user (s) and by the actuators of the system.

The current technical approach for construction of computational models for sending signals to actuators is based on experimentally integrating logical and evolution models of the physical environment. The *inefficiency* in requiring experimental verification and validation that the safety, security, and functional system requirements are satisfied is a *fundamental barrier* to lowering the costs of integrating existing complex information systems. Previous efforts to improve the integration process have been hampered by the fact that there has not been a mathematical framework for simultaneously analyzing safety, security, and other logical requirements while also considering temporal and spatial constraints. Our creation of the hybrid system state provides the technical foundation to achieve the *unification* of logical and evolution models. Our multiple-agent declarative control architecture provides the analytical framework to simultaneously comply with evolution and logical constraints.

Attainment and maintenance of a computable model is accomplished through analysis of the hybrid system state. The Kohn-Nerode approach for unification of logical and evolution models is based on introducing the idea of continuity of the hybrid state representation. The continuity argument and the constructive extraction of automata which comply with the continuity constraint is accomplished by using the mathematics of manifolds (see Figure 2). A

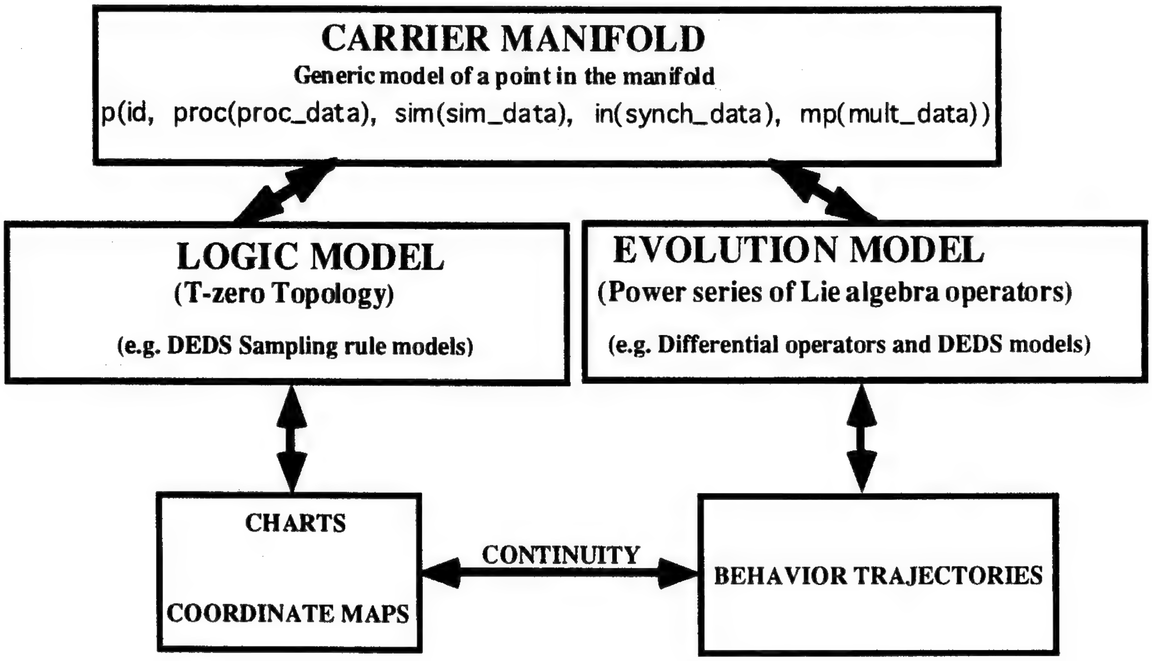

CARRIER MANIFOLD
Generic model of a point in the manifold
p(id, proc(proc_data), sim(sim_data), in(synch_data), mp(mult_data))
LOGIC MODEL
(T-zero Topology)
(e.g. DEDS Sampling rule models)
EVOLUTION MODEL
(Power series of Lie algebra operators)
(e.g. Differential operators and DEDS models)
CHARTS
COORDINATE MAPS
CONTINUITY
BEHAVIOR TRAJECTORIES

Figure 6. Generic model of a point in the carrier manifold

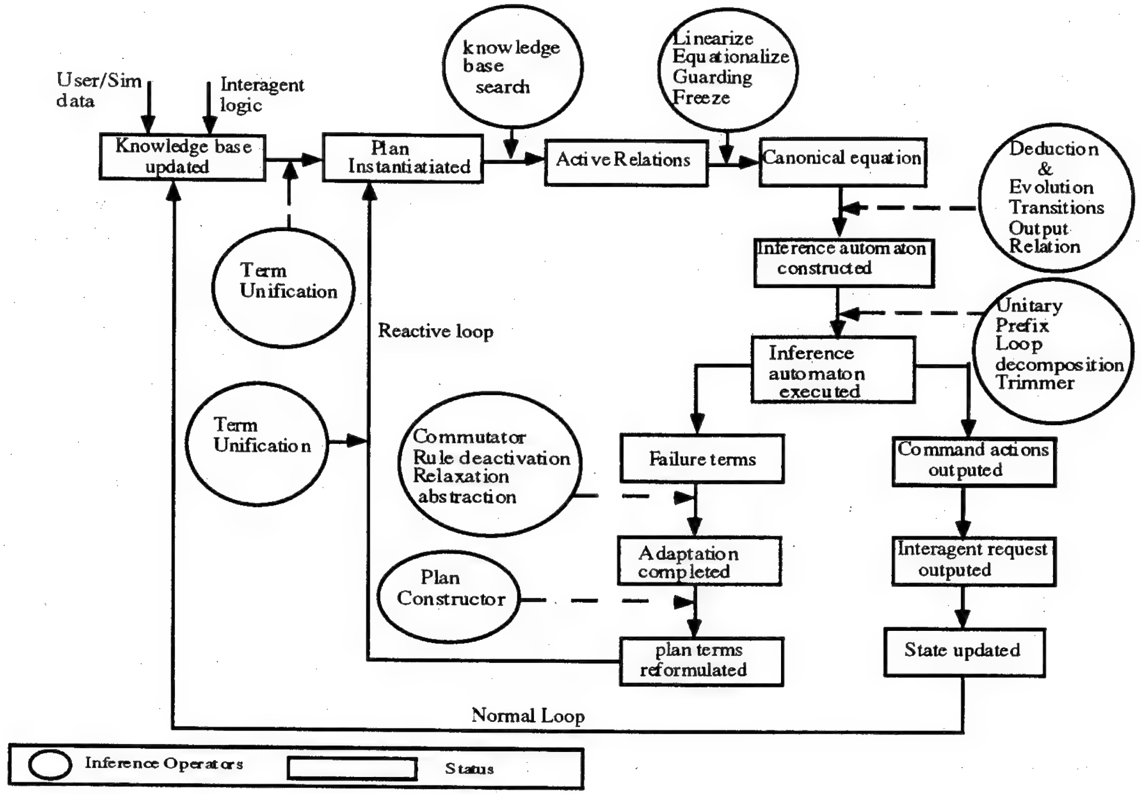


Figure 3. Data Flow Model of Points in the Carrier Manifold

point in a manifold supports unification of logic and evolution models. T-zero topologies have a one-to-one correspondence with horn clauses of logical representations. This enables us to model the Discrete-Event Dynamic System (DEDS) sampling rule models. Lie algebra results concerning infinitesimal operators on smooth functions allow us to consider all the standard evolution models of differential operators and DEDS evolution models. We embed logical models in continuous models in order to construct automata which comply with logical and continuum constraints. Details of the generic unified model are given separately. The data flow model of points in the carrier manifold is given in Figure 3. We assert and emphasize here that for systems which meet the conditions for creation of a hybrid system state, the revolutionary nature of our approach has two benefits:

- Creation of a unified mathematical foundation for analysis and synthesis of models which for decades have been treated separately, and
- Creation of a rigorous process for incremental expansion of trusted systems which must comply with stringent safety and performance constraints.

2. A New Approach to Aggregation and Disaggregation of Components

Agent characteristics: Each agent of the declarative control architecture operates as a real-time theorem prover in the domain of relaxed variational theory developed by L. C. Young [13]. A *customized* version of this theory, enriched with elements of differential geometry and equational logic provides a general representation for the dynamics, constraints, requirements and logic of complex computer-controlled systems.

We will provide an overview of a generic hybrid system architecture and provide a domain reference architecture for target engagement.The agents are connected via an inter-agent network and the effect of each agent in the global network on a local agent is obtained through a global-to local transformation. Inter-agent specification clauses characterize constraints on the relaxed Lagrangian optimization problem. Specifically, they express the constraints imposed by the rest of the network on each agent . They also characterize the global-to-local transformations and local-to-global transformations (see [45]). Finally, they provide the rules for building a generalized multiplier for incorporating the inter-agent constraints into a complete unconstrained criterion. The multiplier and transformations are expanded in rational power series in an algebra discussed in [46].

The conjunction of equational forms for each global-to-local transformation is constructed by applying the following invariant embedding principle:

"For each agent, the actions at given time t in the current interval are the same actions computed at t when the formulation is expanded to include the previous, current, and next intervals."

By transitivity and convexity of the criterion, the principle can be analytically extended to the entire horizon. The invariant embedding equation has the same structure as the dynamic programming equation given in, but with the global criterion and global Hamiltonians instead of the corresponding local ones.

The local-to-global transformations are obtained by inverting the global-to-local transformations, obtained by expressing the invariant embedding equation, as an equational theorem. These inverses exist because of convexity of the relaxed Lagrangian and rationality of the power series.

3. Conclusion: We have provided an overview of the theory of multiple-agent hybrid control. We are currently implementing a multiple-agent hybrid control architecture for the target engagement process.

[1] Nerode, A. and Kohn W." Multiple Agent Declarative Control Architecture" Proc. of the workshop on Hybrid Systems, Lygby, Denmark, Oct 19-21, 1992.

[2] Nerode, A. and Kohn W." Foundations Of Hybrid Systems" In Hybrid Systems, Nerode, A, R, Grossman Eds. Springer Verlag series In Computer Science #726, New York, 1993.

[3] Kohn W., and Nerode A., "Multiple-Agent Hybrid Systems" Proc. IEEE CDC 1992, vol 4, pp 2956, 2972.

[4] Kohn, W. "declarative Control Architecture" CACM Aug 1991,Vol34, No8.

[5] W Kohn, Nerode A. " An Autonomous Systems Control Theory: An Overview" Proc. IEEE CACSD'92, March 17-19, Napa, Ca.,pp 200- 220.

[6] Kohn W,, and Nerode A. "Models For Hybrid Systems: Automata, Topologies, Controllability, Observability" Tecnical Report 93-28, MSI, Cornell University, June,1993.

[7] Garcia, H.E. and A. Ray "Nonlinear Reinforcement Schemes for Learning Automata" Proceedings of the 29th IEEE CDC Conference, Vol. 4, pp 2204–2207, Honolulu, HA, Dec. 5–7, 1990.

[8] Kohn W. and Nerode A. "multiple Agent Hybrid Control Architecture" MSI Report 93-11 Cornell U.

[9] Kohn, W. "Declarative Hierarchical Controllers" Proceedings of the Workshop on Software Tools for Distributed Intelligent Control Systems, pp 141–163, Pacifica, CA, July 17–19, 1990.

[10] Kohn, W. and T. Skillman "Hierarchical Control Systems for Autonomous Space Robots" Proceedings of AIAA Conference in Guidance, Navigation and Control, Vol. 1, pp 382–390, Minneapolis, MN, Aug. 15–18, 1988.

[11] Kohn, W. "A Declarative Theory for Rational Controllers" Proceedings of the 27th IEEE CDC, Vol. 1, pp 131–136, Dec. 7–9, 1988, Austin, TX.

[12] Kohn W. " Multiple Agent Inference in Equational Domains Via Infinitesimal Operators" Proc. Application Specific Symbolic Techniques in High Performance Computin g Environment". The Fields Institute, Oct 17-20 1993.

[13] Young, L.C. "Optimal Control Theory" Chelsea Publishing Co., NY, 1980.

[14] Padawitz, P. "Computing in Horn Clause Theories" Springer Verlag, NY, 1988.

[15] Kohn W. "Multiple Agent Hybrid Control" Proc of thhe NASA-ARO Workshop on formal Models for Intelligent Control, MIT,sept 30-Oct2, 1193.

Simulation as a Tool for Hybrid System Design *

John Lygeros, Datta Godbole & Shankar Sastry

Intelligent Machines and Robotics Laboratory
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720
lygeros,godbole,sastry@eecs.berkeley.edu

Abstract

A case study of the use of simulation as a tool for design and validation of hybrid systems is presented. We use the Intelligent Vehicle Highway Systems (IVHS) architecture of [1], a system that involves both continuous state and discrete event controllers as our example of a hierarchical hybrid system. We point out that even though analytical methods do not exist for verification of hybrid control system, a simulation tool can be useful to (in)validate that the the hybrid system operates properly.

1 Introduction

The term hybrid system has been used to describe a large and rich class of dynamical systems (see for example [2]). A typical system of this class is arranged in a hierarchy of two (or more) layers (Figure 1). At each layer the system is modeled at a different level of abstraction: the lower layer usually contains the physical plant and the low level controllers and is described in terms of differential and/or difference equations while in the higher layers the description is more abstract. Typical choices of descriptive language for these higher layers are finite state machines, fuzzy logic, Petri nets, etc. Clearly an interface is needed to establish communication between different layers. The interface typically plays the role of a translator between signals in the lower layer and symbols in the higher.

In a general hierarchical structure more than two levels may exist. Usually, as we move up the layers, the system description becomes more abstract (i.e., closer to linguistic), information gets condensed (i.e., a symbol at a higher level encodes many condensed facts about the lower levels) and the commands become more descriptive (i.e., a single command at a higher level induces many actions at the lower levels).

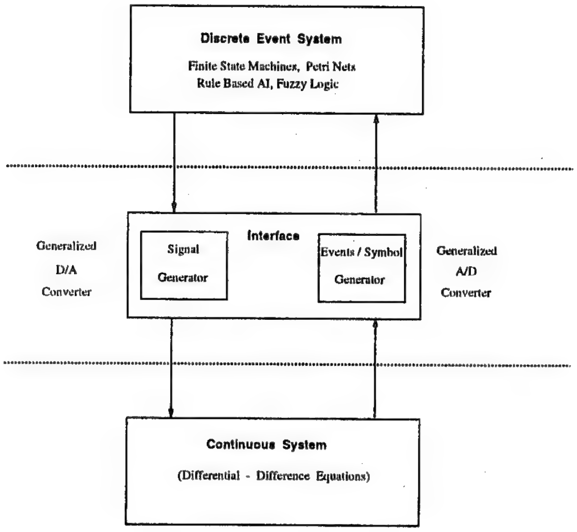

Discrete Event System
Finite State Machines, Petri Nets
Rule Based AI, Fuzzy Logic
Interface
Signal Generator
Events / Symbol Generator
Generalized D/A Converter
Generalized A/D Converter
Continuous System
(Differential - Difference Equations)

Figure 1: Hybrid System Architecture

Designing controllers in such a multilayered environment and analyzing the performance of the resulting closed loop system is a formidable task. Years of research have produced powerful techniques for designing controllers at the individual layers. Standard designs include linear and nonlinear control techniques for the lower layer (e.g., optimal, adaptive and robust control) and supervisory ([3]), fuzzy or other controllers for the higher layers. There is, however, a gap between these techniques: there are no tools for predicting and analyzing the performance of the architecture obtained when the individual layers are brought together. One would expect that if the controllers for the individual layers were designed properly and the

*Research supported in part by the PATH program, Institute of Transportation Studies, University of California, Berkeley, under MOU-135 and ARO under DAAL03-91-G191

0-8186-6440-1/94 $04.00 © 1994 IEEE

architecture was formed using appropriate interfaces the overall system would behave predictably. There are however quite a few examples that suggest that this need not be the case.

Because of this lack of tools, simulation plays a very important (if not indispensable) role in the design of complex, hybrid systems. Even though simulation can not replace formal proof techniques (analytical or computational) it can still provide valuable information about the system performance. More specifically, successful results under extensive simulation indicate that the design is likely to behave well, even though, usually there is still a lot of room left for situations where the system behaves poorly. On the other hand, unsatisfactory performance on the simulation testbed indicates that the design is not good enough for certain cases and may suggest improvements that will eliminate these shortcomings. In other words, simulation results can not be taken as proof that a system works well in general but they can be taken as proof that it works in specific cases, or, more importantly, that it doesn't work in others.

There has been extensive work on the development of techniques for simulating general hierarchical hybrid systems (see e.g. [4] and [5]). Our work is based on a specific simulation package, *SmartPath*, developed for simulation of automated vehicles in an IVHS. Using the control architecture of IVHS as an example, we will demonstrate the role of simulation in design and validation of hybrid systems.

2 Intelligent Vehicle Highway System

An example where the hybrid system structure can be found is an Intelligent Vehicle Highway System (IVHS). The goal is to design a system that can significantly increase safety and highway capacity without having to build new roads, by adding intelligence to both the vehicles and the roadside. In order to achieve this, the notion of "platooning" is introduced. It is assumed that traffic on the highway is organized in groups of tightly spaced vehicles, called platoons. Successful implementation of such a scheme would not only result in substantial increase in capacity (as high as four times the current capacity), but it will also enhance passenger safety. By having the vehicles within a platoon follow each other with a small intra-platoon separation of about 1 meter, we guarantee that if there is a failure and an impact is unavoidable, the relative speed of the vehicles involved in the collision will be small, hence the damage will be minimized. The inter-platoon separation, on the other hand, is large (of the order of 30 meters) to physically isolate the platoons from each other.

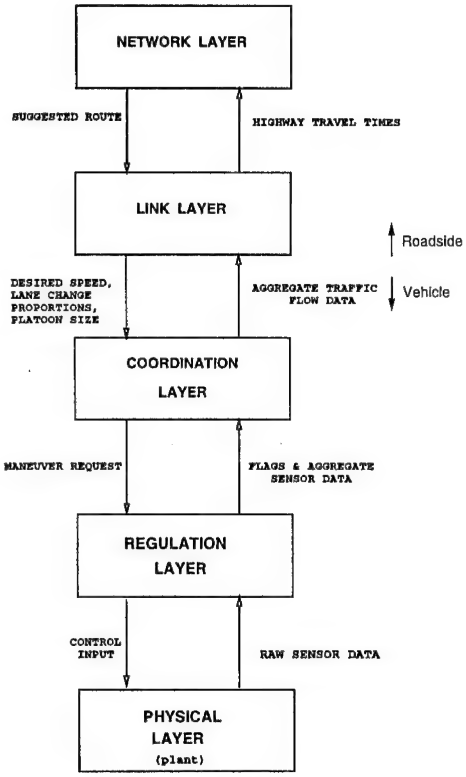


Figure 2: IVHS Architecture

2.1 IVHS Control Architecture

Clearly implementation of such a scheme would require the vehicles to be automatically controlled, as human drivers are not fast and reliable enough to be able to form platoons. In the architecture outlined in [1] the system is organized in five layers (Figure 2).

The top layer, called the **network layer**, is responsible for the flow of traffic on the entire highway system. Its task is to prevent congestion and maximize throughput by dynamic routing of traffic along the interconnected network of highways.

The second layer, called the **link layer**, coordinates the operation of whole sections (links) of the highway. Its primary concern is to maximize throughput while maintaining safe conditions of operation. With these criteria in mind, it calculates an optimum platoon size and an optimum velocity for each highway section. It also decides which lanes the vehicles should follow to get to their destination as fast as possible. Finally, it monitors incidents on the highway and diverts traffic in order to minimize the impact of the incident on traffic flow and safety. The link layer design is based on a fluid flow traffic model, developed and implemented in simulation by Bobby Rao. It uses aggregate statistical information about the traffic in a section. As a consequence the commands it issues are also in aggregate form and are addressed to all the vehicles in the link. A typical command might be "30% of the vehicles going to the next exit change lane now". [6] provides details of a possible link layer design.

The next level of hierarchy below the link layer is the **coordination layer**. It's task is to coordinate the operation of platoons with their neighbors. Coordination layer controller uses three basic maneuvers (join, split and lane change) to organize traffic in platoons. Only a leader or a free agent can initiate a maneuver; a follower needs to request the leader to initiate a maneuver for it[1]. In *join* a leader of the platoon joins the platoon in front to form a larger platoon; in *split* a leader splits the platoon into two at a designated position; in *change lane* a free agent changes lane. The coordination layer controller receives the link layer commands and translates them to specific maneuvers that the platoons need to carry out. For example, it will ask two platoons to join to form a single platoon whose size is closer to the optimum or, given a command like "30% of the vehicles going to the next exit change lane now", it will decide which vehicles will comprise this 30% and split the platoons accordingly in order to let them out. The design of [7] uses protocols, in the form of finite state machines, to organize the maneuvers in a systematic way. They receive the commands of the link layer and aggregated sensor information from the individual vehicles (of the form "there is a vehicle in the adjacent lane"). They then use this information to decide on a control policy and issue commands to the regulation layer. The commands are typically of the form "accelerate to merge to the preceding platoon" or "decelerate so that another vehicle may move into your lane ahead of you".

Below the coordination layer in the control hierarchy lies the **regulation layer**. Its task is to receive the coordination layer commands and translate them to throttle, steering and brake input for the actuators on the vehicle. For this purpose it utilizes a number of continuous time feedback control laws (See [8] and references therein) that use the sensor readings to calculate the actuator inputs required for a particular maneuver. The regulation layer occasionally needs to communicate with the coordination layer to inform it of the outcome of a maneuver.

The bottom layer is not part of the control hierarchy. It is called the **physical layer** and it contains the actual plant (in this case the vehicles with their sensors, actuators and communication equipment and the highway topology). For the purposes of simulation it can be assumed that the physical layer contains models of the actual physical quantities. From a hybrid systems point of view, the physical layer is merged with the regulation layer, as they both use ordinary differential equations to describe the system.

2.2 Hybrid System Issues

From the outline presented above, it is clear that the proposed IVHS control architecture fits nicely into the hierarchical hybrid control framework described in the introduction. The design process and the analysis of the proposed hierarchy leads to observations that relate to fundamental properties of hybrid systems.

2.2.1 Distributed vs. Centralized Control

A key feature of the proposed design is the fact that the vehicles operate as semi-autonomous agents. Each one has its own control objectives and implements its own strategies, but at the same time cooperates with its neighbors and the roadside in an attempt to optimize the performance of the overall system.

Distributed, decentralized decision making has many advantages in this case. For one it greatly simplifies the work of the designer as it provides a way of managing the complexity of the system. The volume of the information that needs to be processed and the commands that have to be issued is such that it is almost impossible to come up with a centralized design. The complexity is further increased by the fact that the vehicles need not have identical characteristics. Therefore a decentralized controller, that deals with vehicles individually, will be easier to design, debug and trouble shoot. In addition to this a decentralized design will provide more gradual degradation of performance in the presence of faults. On the other hand, a centralized design is likely to be more efficient as a centralized controller has access to more information and is therefore better suited to assess the overall system performance. Clearly a compromise between the two approaches has to be reached.

[1] Organization of traffic in platoons implies that, at any moment of time, an automated vehicle is either a *leader* (lead vehicle of a platoon), a *follower* or a *free agent* (single vehicle platoon).

This compromise between centralized and decentralized decision making seems to be a feature of all systems where efficient utilization of a scarce resource is needed. In our example the resource is the freeway. Other examples of such resources include the airport runways in the case of air traffic control and the distribution grid in the case of power systems.

2.2.2 Information Flow

The amount of information that is available at each part of the hierarchy should be determined by the information that a given layer needs in order to make a "good" decision. Each layer needs some minimum amount of data to distinguish a good (e.g. safe or efficient) strategy from a bad one, in a give situation. The information that is provided should be kept as close as possible to this minimum to avoid swamping the higher layer controllers with unnecessary data.

In addition to specifying what information is needed at each layer the designer also has to decide how this information is coded. The choice of descriptive language for the layers of the hierarchy dictates in what form the information should be passed to the various subsystems of the design. In the IVHS example real numbers (raw sensor data) is used in the physical and regulation layers, discrete events are used in the coordination layer and statistical distribution of the traffic are used in the link layer. In order to optimize the flow of information the designer may need to come up with elaborate quantization and coding schemes for each descriptive language.

2.2.3 Interface Issues

Closely related to the information flow is the design of an interface between the various layers. Because the various layers of the hierarchy use different descriptive languages an interface is needed to formalize their interaction. The design of the interface should be done in a way that facilitates the verification of the combined system.

In our example two interfaces are needed, to provide interaction between the coordination, regulation and link layers. [9] describes a possible design for the coordination-regulation interface. The proposed interface is a finite state machine whose transitions depend upon the commands from the coordination layer, the readings of the sensors (physical layer responses) and the state of the continuous controllers. It plays a dual role. On the one side it acts as a symbol to signal translator and therefore directly influences the evolution of the continuous system. It receives the coordination layer commands (symbols) and uses them to switch between the different continuous layer controllers (signals). In addition it keeps track of which of these controllers needs to be initialized (symbol) and carries out this initialization by directly changing the controller state (signal). In the other direction the interface acts as a signal to symbol translator. It processes the sensory information (signal) and presents it to the coordination layer in an aggregate form compatible with the finite state machine formalism (symbol). It also monitors the evolution of the continuous system (signal) and decides if the maneuver in progress is safe or not. If at any stage the maneuver becomes hazardous it aborts it, notifies the coordination layer of its decision (symbol) and switches to a different continuous control law that will get the system back to a safe configuration.

2.2.4 Verification and Validation Issues

Despite the fact that design and verification techniques are available for the individual layers of the hierarchy there are still no tools for verifying the overall design. The bounds on the physical layer capabilities imply limits of the disturbances that the regulation layer can tolerate. For example, even though the control law for the leader of a platoon is designed to maintain "safe" following distance from the vehicle ahead, it can not safeguard against arbitrary disturbances, because the vehicle capabilities (e.g. acceleration bounded by [-5,2] m/s^2) and sensor ranges (e.g. longitudinal distance sensor range is 60m) are limited. The controller can guarantee that there will be no accident provided that the vehicle ahead does not start too close and decelerate too fast at the same time. For complete verification therefore we need to guarantee that the disturbances at the lower layers due to the activity of the higher layers does not go outside these limits.

Unfortunately the current theory does not support such verification techniques. Moreover, the cost of building and maintaining a prototype for the combined system is so great that it makes experimental verification impractical. Therefore the only possible way of testing the design is by simulation. The next section describes how this idea was applied to the IVHS problem.

3 SmartPath Simulation

In order to test the performance of the combined system, a dedicated simulator, called SmartPath [10], was developed. The core of the simulator was developed by Farokh Eskafi and its capabilities were subsequently extended by other researchers. The simulation language CSIM was used as the backbone for the platform. It provides all the necessary message passing between and within the layers of the architecture. It also provides a convenient way of coordinating the operation of functions that need to operate asynchronously and at a different time scale. SmartPath can be classified as a microscopic traffic simulator, because it simulates the dynamics of each vehicle individually.

3.1 Simulation Setup

Starting from the bottom of the hierarchy, the physical layer was implemented as a C function coding the differential equations that describe the dynamics of each vehicle. This plant was integrated using a fourth order, variable step Kutta-Merson routine. Therefore the physical layer essentially operates in continuous time.

The regulation layer was also implemented in C. It consists of a number of functions that calculate the necessary inputs to the physical layer, namely the throttle, brake and the steering input to the vehicle. The operation of all these functions is coordinated by means of an additional, supervisory function that acts as an interface between the regulation and coordination layers. The controllers coded in SmartPath were designed in continuous time and then sampled every 5 ms, a period dictated by the sampling frequency response of the actuators that are in use on the real vehicles.

The coordination layer protocols were implemented in CSIM. The time frame used here is even slower: the coordination layer interacts with the regulation layer every 100ms. This value is dictated by the sampling frequency response of the communication devices and the sensors.

Finally, the fluid flow algorithms that specify the behavior of the link layer were implemented as C functions. Compared to the other layers the rate at which they operate is very slow. New commands are issued roughly every 25 seconds of simulation, depending on the velocity of the vehicles and the length of the highway section in question.

The organization of the simulation code mimics the hierarchical structure of the controller design. Individual modules for the regulation feedback controllers, the coordination maneuver protocols and the link algorithms, as well as sensors, actuators and communication devices exist separately. Their interactions are determined by the corresponding interactions of the theoretical design. This approach may be slightly inefficient in terms of memory requirements and length of code, but it facilitates the understanding of the code and the implementation of modification, whenever the design is changed. Moreover it allows us to incorporate interesting effects to our simulation, such as sensor noise, actuator dynamics and external disturbances (e.g. longitudinal and lateral wind). Finally the modular arrangement highlights interesting facts about the hybrid nature of the problem, such as the information flow, the interface requirements, etc.

3.2 Simulation Results

The development of the simulation platform was carried out in two stages. First the individual layers were implemented and tested separately. The results of these tests demonstrated that the behavior expected from the formal analysis was indeed obtained. For example the regulation layer control laws (like the one carried out by the leader of a platoon) were simulated one at a time and they all proved to be stable and display desirable properties. Similarly the coordination layer protocols were simulated using a very simple abstraction to model the regulation and physical layers. They also worked well, as expected by the results of the automatic verification. Finally the link layer design was also tested and tuned using a different simulation program, called *SmartLink* (developed by Bobby Rao). SmartLink carries out a macroscopic simulation of the highway as it codes the fluid flow traffic model directly, rather than simulating individual vehicles.

At a second stage the components were combined using appropriate interfaces and the overall system was simulated. After many long simulation runs the results were very encouraging; the system behaved well in most cases. There were however a few situations where unacceptable behavior (e.g. high speed collisions) was observed. These situations were not predicted by the off-line analysis carried out at the individual layers. The simulator however allowed us not only to observe them but also to determine their causes. Using this information the control design has been modified to iron out these problems and improve the performance of the system. We now describe some of the observed collisions.

Lane change across speed differential
According to the coordination layer design, only free agents (single vehicle platoons) are allowed to change lane. Before a vehicle initiates a lane change it looks (through its sensors) to the adjacent lane to make sure

that there is room for it there. If no vehicle is visible in the sensor range the move is initiated immediately. If a vehicle is found and its distance is less than the safe inter-platoon spacing, communication is established to coordinate the maneuver. This goes on until a gap twice as large as the safe inter-platoon spacing is found. Then the lane change takes place in the middle of this gap.

In most situations this arrangement should cause no problems. Indeed both the protocol that coordinates the maneuver and the regulation layer controllers that align the free agent with a gap in the next lane have been proven to perform well. However, consider the following scenario. Free agent A switches from a slow lane to a fast lane. During the change a gap big enough for A to move into is present in the fast lane (for example no vehicle is visible in the sensor range). It is conceivable however that a vehicle (denoted by B) is present in the fast lane behind A, which, after the lane change is complete, finds itself just outside A's rear sensor range (say $35m$) and moving a lot faster than A (say $30m/s$ as opposed to $10m/s$). It turns out that the AICC lead controller is incapable of recovering from such drastic initial conditions, so a crash is inevitable. Similar crashes have been observed crashes during lane change from a fast to a slow lane.

Joining a decelerating platoon
The feedback control law for 'join' maneuver is designed in two steps in [8]. In the first step, an open loop trajectory is calculated based on the assumption that the platoon in front will travel at constant velocity during the maneuver. In the second step, a feedback control law is used to asymptotically track this desired trajectory. With this control law, changes in the velocity of the front platoon should cause no problem as the state feedback should take care of any deviations from the desired trajectory. However, the actual trajectory *will* deviate from the desired one if the limits of the actuators (throttle and brake) are reached. To avoid this possibility, the interface aborts the maneuver when it detects the danger of actuator saturation (see [9]). After aborting the maneuver the system should find itself in a position from which it can continue safely under the AICC lead control law. The simulation indicates that under extreme conditions this may not be true and 'join' maneuver may cause a major hazard.

Such large decelerations were created in the simulation when we asked all cars in a section of one lane to exit at the same time. This caused a number of splits in the platoons as the vehicles tried to become free agents in order to change lanes. The deceleration built up enough to cause saturation of the actuators. Therefore vehicles undergoing 'join' maneuver upstream of the disruption were forced to abort their maneuvers. In this case, the vehicles that aborted join found themselves moving faster than the platoon ahead and closer than desired safety distance. The extreme decelerations saturated the actuators and thus caused crashes.

3.3 Analysis of the simulation results

The crashes described above illustrate the importance of data abstraction, interface design and information flow in hierarchical and distributed control systems. The common feature of all the crashes is that they are caused by continuous layer performance not accounted for by the discrete layer. In all cases this leads to the discrete layer making requests that are incompatible with the current situation of the continuous layer, such as a potentially dangerous maneuver.

Apart from fixing the problems encountered here, these observations naturally lead to the question to what extend can one trust the conventional discrete and continuous verification techniques when it comes to hybrid systems. Clearly if any faith is to be placed in the IVHS design presented here a proof of its performance claims is needed. This example indicates that such a proof is not possible using conventional tools.

3.4 Additional Features

To enhance the capabilities of the basic simulation platform and to make it more user friendly, some additional features were implemented. From the first stages of the development the simulator was coupled to an animation program, developed by Delnaz Khorramabadi. Typically a long simulation is executed, the necessary information is stored and then a movie of the vehicles as they move along the freeway is displayed. Among other things, the animation package allows the user to modify the viewing position, to track a vehicle, to monitor the communication messages that are exchanged and to obtain information about vehicle state. These features proved very useful, especially in identifying the causes of high speed collisions, as they allowed us to visualize the traffic conditions rather than try to infer what is going on from the state trajectories of individual vehicles. Furthermore, building on the capabilities of the animation, an interactive version of the simulator was developed. For this the simulation and animation are executed simultaneously. The user is allowed to select a vehicle using the animation display and to force it to carry out various maneuvers, like accelerate, decelerate and change lane. This allows us to introduce disturbances to the system and investigate how the control architecture responds to abnormal conditions. We are currently working on extending the simulation capabilities to include various abnormal conditions on the highway (e.g. rain, poor visibility, sensor faults etc). Together with the controller for fault handling this will be useful in obtaining qualitative and quantitative measures of safety (e.g. number of accidents per highway miles) on the automated highway.

Ultimately we would like to be able to use this simulation package to investigate the behavior of a large scale, multihighway system, for example the entire highway system of the San Francisco Bay Area. Because of the number of vehicles involved and the fact that each vehicle is simulated independently, a simulation of this scale will be very time consuming. To solve this problem, work is currently underway to produce a parallel version of the SmartPath simulator that will run on a CM5 connection machine. At the same time an effort is made to couple SmartPath with an object oriented data base. This will add a lot of flexibility to the program, as it will make it easy to change the highway configuration, add new types of vehicles, change the control laws and/or the control architecture and store the state of the system so that simulations can be stopped and restarted at any point.

4 Conclusions

In this paper, we have shown that simulation can be a very useful tool in validation of hierarchical hybrid dynamical systems. We have shown that independent verification of discrete and continuous layers may not always guarantee satisfactory performance of the hybrid system. The problem arises because of the fact that the discrete layer can only comprehend abstractions of the continuous layer. As a result it can not take sufficiently into account certain continuous layer parameters, which in our case were the sensor and actuator ranges and the controller performance bounds. These observations led to the conclusion that in order to fully trust the design we need verification tools that test the performance of the combined *hybrid* system. Until such tools are available simulation can be used to replace them in the verification/redesign process for hybrid control systems.

Acknowledgement:

The authors would like to thank Farokh Eskafi, Delnaz Khorramabadi, Dr. Bobby Rao and Prof. Pravin Varaiya for helpful discussions providing insight into the problem.

References

[1] P. Varaiya, "Smart cars on smart roads: problems of control," *IEEE Transactions on Automatic Control*, vol. AC-38, no. 2, pp. 195–207, 1993.

[2] R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, *Hybrid Systems*. Springer Verlag, 1993.

[3] P. J. G. Ramadge and W. M. Wonham, "The control of discrete event dynamical systems," *Proceedings of IEEE*, vol. Vol.77, no. 1, pp. 81–98, 1989.

[4] A. Back, J. Guckenheimer, and M. Myers, "A dynamical simulation facility for hybrid systems," Tech. Rep. 92-6, Cornell University, April 1992.

[5] L. Tavernini, "Differential automata and their simulators," *Nonlinear Analysis, Theory, Methods and Applications*, vol. 11(6), pp. 665–683, 1987.

[6] B. S. Y. Rao and P. Varaiya, "Roadside intelligence for flow control in an IVHS," *Transportation Research - C*, vol. 2, no. 1, pp. 49–72, 1994.

[7] A. Hsu, F. Eskafi, S. Sachs, and P. Varaiya, "Protocol design for an automated highway system," *Discrete Event Dynamic Systems*, vol. 2, no. 1, pp. 183–206, 1994.

[8] D. N. Godbole and J. Lygeros, "Longitudinal control of the lead car of a platoon," in *American Control Conference*, pp. 398–402, 1994.

[9] J. Lygeros and D. N. Godbole, "An interface between continuous and discrete-event controllers for vehicle automation," in *American Control Conference*, pp. 801–805, 1994.

[10] F. Eskafi, D. Khorramabadi, and P. Varaiya, "Smartpath: An automated highway system simulator," Tech. Rep. PATH Technical Note UCB-ITS-94-4, Institute of Transportation Studies, University of California, Berkeley, 1994.

Session 1B:

Modeling and Dynamics

Expressing intratask parallelism in discrete event simulation models

Ashvin Radiya
Department of Computer Science
The Wichita State University
Wichita, KS 67260
radiya@cs.twsu.edu

Abstract

For parallel and distributed simulation, a model is usually defined to consist of independent tasks which synchronize by communicating time-stamped events/messages. In this paper, we define a discrete event simulation modeling paradigm which supports explicit expression of intratask parallelism, i.e., parallelism within a task. In this paradigm a task is defined by a set of rules whose actions are triggered based on multiple simultaneous event occurrences. New ways of expressing parallelism in a model emerges because (1) actions of multiple rules can be executed in parallel and (2) an action of a rule can be defined by data parallelism on parameter values of simultaneous event occurrences. The usefulness of the constructs of our paradigm is illustrated by an example.

1 Introduction

The limitations of even the best of compilers in extracting parallelism from sequential programs are being increasingly realized. For complex practical programs, it is often the case that a programmer can write efficient parallel programs more easily than a parallelizing compiler. Hence, it is important to develop parallel programming paradigms and languages in which parallelism can be explicitly expressed by the programmers. In a good parallel programming language, a problem can not only be solved more efficiently but better solutions can be developed because the parallelism inherent in the problem is expressible directly in the programming language. In contrast, if the programming language is sequential then the inherent parallelism in the application known to the programmer has to be translated into a sequential code by the programmer which may introduce subtle logical errors. The parallelism expressed directly by a programmer is referred to as *explicit parallelism* and the parallelism not directly expressed but extracted by a (compiler) technique is referred to as *implicit parallelism.* This paper defines a new modeling paradigm that facilitates expression of explicit parallelism in discrete event simulation models.

The key to expressing explicit parallelism (henceforth simply referred to as parallelism) in discrete event simulation models is to identify independent computations around the dimensions of state and time. The concept of task, process [Mi86, Fu90], or component [Ze84, Ze90] provides a basis for defining independent computations around the dimension of space. Each task has its own state and interacts with other tasks by exchanging events (also called signals or messages). A task is like a black box in the sense that its state is not visible from the outside. Hence, tasks can be distributed on separate processors during simulation, and interactions among tasks are captured through message passing. This gives rise to intertask parallelism in which actions modifying states of different tasks can be executed in parallel. A language in which models can be defined as tasks is said to *support expression of explicit intertask parallelism.*

The second dimension of time is critical to simulation modeling. Many concepts such as event, activity, and process have been developed to express time dependent computations. Time related behavior is expressed by associating simulation clock time with *entities* such as events, activities, processes, messages, etc. In the underlying simulation procedure, a list, called Future Entity List (FEL) of scheduled entities is maintained. The simulation proceeds by removing an entity with the earliest time from FEL and executing some code based on the removed entity. The processing of an entity may insert new time-stamped entities and/or remove existing entities from FEL. If there are two or more entities with the same earliest time then the order in which they are processed is important because the order may affect the simulation results. The order in which entities with the

0-8186-6440-1/94 $04.00 © 1994 IEEE

same time are processed is controlled by the modeler by assigning priorities to entities.

It is possible that two or more entities from FEL with the same time, or even different times, can be processed in parallel. The parallel simulation community has mainly focused on developing and studying techniques such as conservative and optimistic simulation strategies [Mi86, Fu90] which extract parallelism from models whose basic semantics is sequential as defined above. In our categorization defined above, these techniques extract *implicit parallelism.* As it is difficult to automatically extract parallelism from a model, language constructs have also been developed for giving hints to the compiler by the modeler. However, these language constructs and parallel simulation strategies do not change the essential world view of the modeler who is still required to define a sequential model of inherently parallel system. Moreover, now the modeler is required to give hints to compiler for parallelization. This requires the modeler to understand the behavior of the system and the compiler.

In the literature some parallel simulation languages such as Maisie [BL90] have been developed. However, these languages allow the modeler to explicitly express intertask parallelism but not intratask parallelism, i.e., the parallelism within a task. Typically, a task is defined by an event scheduling model in which one routine (method or procedure) is defined for each type of event (message). The modeler defines a model by viewing that one event is removed at a time and its routine is processed. The parallelism within a task is implicitly extracted by the simulator using the optimistic or conservative simulation strategies.

In this paper, we define a modeling paradigm in which intratask parallelism can be explicitly expressed. The modeling paradigm requires the modeler to think that all events with the same earliest clock time are removed, and based on the removed events many computations are triggered in parallel.

2 Intratask parallelism

Explicit expression of intratask parallelism virtually remains unexplored in simulation modeling. In this paper, we explore some principles of expressing intratask parallelism in simulation models. In contrast to the intertask parallelism, the elements expressing intratask parallelism share a common state and events. It can be argued that intratask parallelism is not required because one can always divide a task into subtasks and utilize the techniques of intertask parallelism. However, in general this may not be feasible or desirable because a task is a "logical entity", and decomposing it into subtasks may lead to artificial and complex decomposition of the state and events of the task.

Many techniques have been developed for expressing intratask parallelism in the parallel programming literature. One technique is to define a task as an interleaving computation where different "threads" synchronize using devices such as shared memory, memory locks, and/or semaphores. Another important technique is to express the data parallelism available in a task as a Single Program Multiple Data (SPMD) program. These and other such techniques can be applied for expressing parallelism in simulation models as well. However, the applications of these techniques in simulation is limited by the structure of models. The structure of models defined in the modeling paradigm developed in this paper facilitates applications of the above parallelization techniques in a broader context.

A task in a simulation model is different from a task in a nonsimulation program due to the fact that events and computations in simulation are stamped with simulation clock time. Hence, it is possible that more than one event of the same or different type may occur at an instant. For example, many events of type *arrival(priority)* may occur at an instant corresponding to simultaneous arrival of customers. The approach to handling simultaneous event occurrences is affected by how a task is defined. The common approach of defining a task is to define one event routine for each type of event. Then, the routines of simultaneously occurred events are executed in the order of priorities assigned to events.

In our approach a task is defined by a set of rules, where each *rule* consists of

a. an event-expression which can refer to multiple simultaneous event occurrences and state conditions, and

b. an action which defines state modifications and schedules events.

The action of a rule gets fired whenever the events referenced in the event-expression of the rule occur at the same point in time. Once the action of a rule is fired, it can refer to parameters of one or more events occurring at the time of firing. This makes it possible to express intratask parallelism in the following two forms.

a. The action of a single rule defines data parallelism on data or parameter values of multiple event occurrences.

b. The actions of all rules whose event expressions are satisfied are fired in parallel.

2.1 Referring to multiple simultaneous event occurrences

Simultaneous event occurrences play an important role in expressing intratask parallelism. The basic approach to defining interactions among simultaneous event occurrences has been defined in the Logic-Based Foundation (*LBF*) of discrete event modeling and simulation [Ra90, RS94]. In *LBF*, events are defined to be logical propositions or relations. For example, *LBF* treats the events *customer_arrives* and *customer_departs* as logical propositions. To claim that the event *customer_arrives* occurs at an instant t is to claim that the proposition *customer_arrives* is true at t. The benefit of this view of event is that interactions among simultaneously occurring events can be defined by logical conditions on events. The conjunction of two events using & means that both events occur together, and the negation of an event using ~ means that the event does not occur. For example, the condition (*customer_arrives* & *customer_departs*) asserts that both events *customer_arrives* and *customer_departs* have occurred. Whereas the condition (*customer_arrives* & ~ *customer_departs*) asserts that the event *customer_arrives* has occurred but the event *customer_departs* has not occurred.

In general, events are parameterized, i.e., events denote various types of relations which are true for one or more tuples of values. For example, multiple occurrences of $arrival(priority, gender)$ may denote the bag $\{(low, male), (low, male), (low, female), (high, female), (high, female)\}$ of values of the parameters *priority* and *gender* corresponding to simultaneous arrivals of two low priority male customers and one low priority and two high priority female customers. This example shows that sets and bags of values are associated with event occurrences. We define three types of event relations as follows.

primitive — at any instant the event relation is true for at most one tuple. Typical examples are *the_president_arrives*, *the_whitehouse_opened*, and *NY_stockmarket_fell*.

set — at any instant the event relation is true for at most a finite *set* of tuples. An example of this type of event is *release_disk*(*disk_no*) denoting the release of disk *disk_no* in a multi-disk computer system. The type of event relation *release_disk* is **set** because at a particular instant it can be true for more than one disk but for each *disk_no* only one event *release_disk*(*disk_no*) can occur.

bag — at any instant the event relation is true for at most a finite *bag* of tuples. A typical example is *customer_arrives*(*priority*) which may be true for more than one value of the parameter *priority* because a *high*- and a *low*-priority customer may arrive at the same instant. It is also possible that more than one customer of the same priority may arrive at an instant.

In the syntax of *LBF*'s language L_{DE}, an *event-term* consists of an event name followed by a list of arguments. An argument can be any expression of the corresponding parameter type, or "_" which indicates "any value". An event-term can occur in two contexts: one is in a set-expression and the other is in a condition. In a set-expression, at a given instant, an event-term denotes a set of tuples such that for each tuple in the set, the event relation is true and the evaluation of arguments of the event-term match the corresponding components of the tuple. For example, at an instant t, the event-term $arrival(high, \_)$ denotes the bag $\{(high, x) \mid arrival(high, x)$ is true at t, where x can be either $male$ or $female\}$, whereas $arrival(\_, \_)$, or simply *arrival* denotes the bag $\{(x, y) \mid arrival(x, y)$ is true at t, where x can be $high$ or low and y can be $male$ or $female\}$. An event-term can also occur in the context of a condition, in which case it denotes the truth value **true** iff the event relation is **true** for at least one tuple at that instant.

2.2 Parallelism in actions

As mentioned before, the behavior of a task is defined by a set of rules, each of which consists of an event-expression and an action. The intratask parallelism arises in two ways — (1) the code of an action can represent parallelism using the denotation of any event and (2) the actions of rules whose event-expressions are satisfied can be executed in parallel. The parallelism in an action can be expressed by the technique of data or control parallelism [RDR94]. The data parallelism is expressed using **for-all** construct

for-all x **in** S A

which executes $|S|$-many instances of action A in parallel, where each instance of action A gets a different value for x from the set denoted by set-expression S. In LBF, event-terms are set-expressions because they denote a bag of tuples of values. As sets and bags play a crucial role in expressing parallelism, we provide a sophisticated type system and powerful set manipulation operations. To allow manipulation of sets, the database set manipulation operators **select**, **project**, **union**, **difference**, and **cartesian product** are provided. The implementations of these set operators are described by Ullman in [Ul88]. In contrast, the data parallelism in parallel programming is usually based on integer arrays, which is also allowed in our framework.

The control parallelism in L_{DE} is expressed by

$$A_1 \parallel \ldots \parallel A_n$$

which executes different actions $A_1, \ldots, A_n$ in parallel.

Note that the explicit parallelism expressed inside an action can refer to parameters of several event occurrences. Such parallelism would be difficult to automatically derive by a compiler from an event-scheduling model.

In LBF, actions are executed in parallel *without* interleaving. Noninterleaving parallel computations are defined in terms of the concept of "modification" which defines the changes that must be made to the state at an instant due to the execution of an action. In particular, the action is not viewed to define define a new state. This allows us to determine modifications corresponding to applications of multiple rules in parallel. An elaborate theory of modifications has been developed which defines operations for combining modifications [RS94].

3 Example

A system of colliding balls (CB) defined in [Fi94] is modeled in L_{DE} to illustrate the concepts defined in the paper. The CB consists of n perfectly elastic tiny balls (molecules) which change their direction of motion upon striking a wall. It is assumed that balls do not collide with each other, and the containing box is two dimensional [Fi94]. Although, the balls are moving continuously, it is desirable to develop a discrete-event model of CB if we are interested in developing a fast, real-time display of the movements of balls, or in counting the number of times the balls hit walls.

The event-scheduling model of CB defined below is based on the event-scheduling model of CB defined in [Fi94]. Unlike in [Fi94], our model utilizes a parameterized event *col*$(bNo, wall)$ to denote a collision of a ball bNo colliding with a wall $wall$. We need to capture the following in a model of CB: every time one or more balls collide with a wall, (1) for *every* ball, update its position and (2) for every colliding ball, update its velocity (hence, change the direction) and schedule the next occurrence of *col*$(bNo, wall)$. However, the fact that more than one ball may collide at the same time makes the event-scheduling model complex because events *col*$(bNo, wall)$ corresponding to the balls colliding at the same time are processed sequentially. The basic idea of the event-scheduling model given below is that only the first of possibly many simultaneously colliding balls updates positions of all balls. Also, each simultaneously colliding ball (including the first) updates the velocity of that ball and the next *col* event for that ball. The following is the pseudo code of an event-scheduling model of CB which consists of *initialization* and *col* event-routines.

Event-scheduling model of CB

```
event-routine initialization
    $PrevEventTime := -1$
    schedule col($bNo$) for each ball

event-routine col($bNo, wall$)
    /* collision of ball $bNo$ on wall $wall$ */

  1. if current_time() > $PrevEventTime$ then compute the time elapsed since last update using collision_time($bNo$) defined below and update positions of all balls using the elapsed time.
     /* The then-part of this conditional statement is executed only by the first of the possibly many simultaneously colliding balls. The time elapsed since last update is computed using ball $bNo$'s position and velocity at the time of last update of positions. Using this elapsed time, positions of all balls including $bNo$ are updated.*/

  2. Update velocity of $bNo$ to account for reflection at $wall$.

  3. Compute $NextCollisionTime$ and $NextCollidingWall$.

  4. schedule col($bNo, NextCollidingWall$) after $NextCollisionTime$

  5. $PrevEventTime$ := current_time().
```

Similar to the above event-scheduling model, only one rule is required in the L_{DE} model of CB defined below. This rule is executed when its event-expression *col* is satisfied, i.e., when one or more balls collide

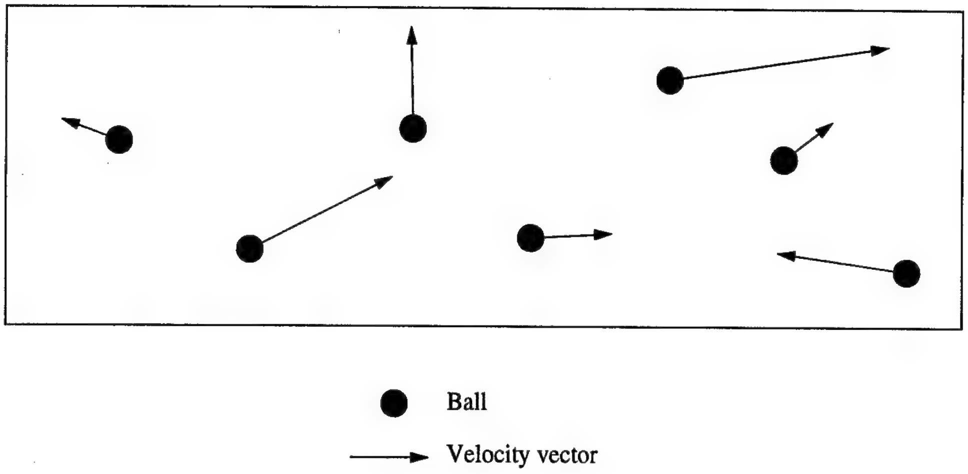


Figure 1: Motion of balls in a two-dimensional box.

walls. In the action of the rule, the technique of data parallelism is used to accomplish the tasks of (1) updating the position and velocity and scheduling next *col* event for each colliding ball in parallel and (2) updating the positions of all noncolliding balls in parallel. The data parallelism is expressed using the **for-all** statement. Furthermore, the **for-all** statements defining these two tasks are connected by ||, and hence they can be executed in parallel. Note that the parallel construct || of L_{DE} defines control parallelism. The control or data parallelism can also be used to define the code of procedures, e.g. see *update_pos*().

An L_{DE} model of CB

Types:

$BALLS = 1..no\_of\_balls;$

$WALLS = \{left, right, top, bottom\};$

Event declarations:

set $col(b \in BALLS, w \in WALLS)$ — *col* is a **set** type of an event which indicates collision of ball b with wall w. *col*'s type is **set** because any two simultaneous occurrences of *col* has different values of parameters, i.e., there is no duplication.

Variables:

B : **set of** $BALLS$ — a set variable to hold a set of balls.

$x[\,]$, $y[\,]$ — arrays to hold x- and y-coordinates of balls.

$vx[\,]$, $vy[\,]$ — arrays to hold x- and y-components of the vectors representing velocities of balls.

Procedures:

procedure *collision_time*(b)
Returns the collision time and the wall of collision for a ball b using b's current position and velocity. The type of this function is $collision\_time(b \in BALLS) \in R^+ \times WALLS$.

```
procedure $update\_pos\_vel(t, b, wall)$
/* The type of this function is $update\_pos\_vel(t \in R^+, b \in BALLS, w \in WALLS) \in$ void. It parallely updates $b$'s position $(x[b], y[b])$ and velocity $(vx[b], vy[b])$ using time $t$ and wall $wall$. */
   $x[b] := x[b] + t \times vx[b]$
|| if ($wall = left$ or $wall = right$) $vx[b] := -vx[b]$
|| $y[b] := y[b] + t \times vy[b]$
|| if ($wall = top$ or $wall = bottom$) $vy[b] := -vy[b]$
```

procedure $update\_pos(t \in R^+, b \in \mathbf{N}) \in$ **void** — updates b's position $(x[b], y[b])$ using time t.

Rules:

```
whenever $col$
   $B := pr_1(col)$;[1]
   for-any  $b$ in $col$
         $\{(impact\_time, \_) := collision\_time(b)\}$;
   {for-all  $(b, old\_wall)$ in $col$
      $\{update\_pos\_vel(impact\_time, b, old\_wall)$;
```

[1] pr_1 is a projection function that returns the set containing first component of tuples in a set (here *col*). The symbol $\ominus$ denotes the operation of difference on sets.

```
    $(t, wall) := collision\_time(b)$;
    schedule $col(b, wall)$ after $t$;}
|| for-all $b$ in $BALLS \ominus B$
    {$update\_pos(impact\_time, b)$}}
```

4 Conclusions

The logic-based foundation (LBF) of discrete event modeling and simulation views that events denote various types of logical relations. This view makes it possible to refer to simultaneous multiple occurrences of events in the conditions which trigger rules and in the actions of rules. The data or parameter values of simultaneously occurring events can be used to express data and control parallelism in the actions of rules. In LBF, in addition to the statements within a single action, actions of different rules in a task can be executed in parallel.

References

[BL90] R. L. Bargodia and W. Liao. *Maisie User Manual.* Computer Science department, UCLA, Los Angeles, CA, 1990.

[CZ94] A. C. Chow and B. P. Zeigler. Revised DEVS: A parallel, hierarchical, modular modeling formalism. Working paper, 1994.

[Fi94] P. A. Fishwick. Simulation model design & execution: Building digital worlds. Prentice Hall, 1994.

[Fu90] R. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10): 33-53.

[Mi86] J. Misra. Distributed Discrete-Event Simulation. *ACM Computing Surveys*, 18, 1, March, 1986, 39-65.

[Ra90] A. Radiya. A Logical Approach to Discrete Event Modeling and Simulation. *PhD Dissertation*, School of Computer and Information Science, Syracuse University, 1990.

[Ra93] A. Radiya. On the expressivity of the *cancel* construct and the temporal operator *unless. The fourth Annual conference on Artificial Intelligence, Simulation*, Planning in High Autonomy Systems, Tuscon, Arizona, Sept. 1993.

[RDR94] A. Radiya, V. A. Dixit-Radiya, and Nimish Radia. Event parallelism and its intergration with Task, Control, and Data Parallelism. Working paper, 1994.

[RS94] A. Radiya and R. G. Sargent. A Logic-based Foundation of Discrete Event Modeling and Simulation. *ACM Transactions on Modeling and Simulation*, Vol. 4, No. 1, January, 1994, 3-52.

[Ul88] J. Ullman. *Principles of Database and Knowledge-Base Systems — Classical Database Systems*: Vol. 1. Computer Science Press Inc, 1988.

[Ze90] Zeigler, B. P., *Object Oriented Simulation with Hierarchical Modular Models*, Academic Press, New York, 1990.

[Ze84] Zeigler, B. P., *Multifaceted Modelling and Discrete Event Simulation*, Academic Press, New York, 1984.

Design of An Efficient Frame-Based Modeling And Simulation Tool

Lien-Pharn Chien

Department of Information Engineering
Kaohsiung Polytechnic Institute
Taiwan

Ray Yueh-Min Huang

Department of Engineering Science
National Cheng Kung University
Taiwan

Abstract

To accurately trace the dynamic behavior of the systems to be developed, and to effectively proceed with the systems' performance measures prior to their implementation, are getting harder as advanced technology is employed in the system design process. This situation drives the functional progress in modern modeling and simulation supporting environment. Our study to the trend is concentrated on promoting the efficiency of modeling and simulating diverse application systems. The achievement in the research is the establishment of a frame-based modeling and simulation tool termed FRAMS. The feature deserved by FRAMS is not just high reuse of model resource but great simulation time saving by modeling an application system with a hierarchical, multilink data structure. Currently, FRAMS has been implemented more specific to performance measures on computer systems. However, the design spirit of FRAMS is for general purposes in extensive application areas.

1 Introduction

Diverse sorts of modeling and simulation environments used in extensive areas for specific or general purposes have been developed within the past few decades. SIMSCRIPT II.5, GPSS and SLAM II are typical examples [1, 7, 8]. Given the existing ones, newer kinds of modeling and simulation supporting environments are still being invented. The major reason to reflect the trend is to promote their processing power and efficiency while new techniques either in hardware or software are explored out. The related improvement primarily includes: 1) to automate the modeling process in order to speed up the ability in modeling an application system being investigated, 2) to apply the object-oriented programming scheme in designing the supporting environment such that models are objects which can be highly reused, and the execution of simulation is carried out by the message-passing method, 3) to shorten the simulation time by distributing the simulation burden in a loosely coupled distributed or closely coupled multiprocessor system, and 4) to adopt an approach of the hierarchical decomposition and hybrid method so that a subsystem can be replaced by an equivalent lumped component [10], and the original larger system model with larger state space is thus shrunk to its equivalent but smaller size having much smaller state space; therefore, the time spent in model simulation is able to be reduced significantly.

As the goal in the research is targeted for performance measures, various sorts of queueing models and queueing networks introduced in the field of Queueing Theory [6] become the targets to be modelled by the proposed modeling and simulation tool. Since these queueing configurations have the trait of random behavior in operation along the time, the simulation time required to execute an modelled system usually takes time before the behavior of the system is in equilibrium. Hence, how to lower down the unavoidable problem is the major concern in the paper. By realizing the point and the evolution of modern modeling and simulation designing methodology, an environment called *FRAMS* (FRAme-based Modeling and Simulation tool) is implemented having the support of a window-driven user interface. Along with *FRAMS*,

0-8186-6440-1/94 $04.00 © 1994 IEEE

an innovative approach utilizing the object-oriented concept but no message-passing feature, and the frame structure [9] is presented. The key idea behind this approach is to avoid the large portion of time spent in message passing between models during model simulation. By using the frame's characteristics to represent objects, and then to form the related *frame bases*, all the frames required to construct a complete system model used in simulation will be organized in a hierarchical, multilink data structure manipulated in a single process. By means of the processing of a single process, the simulation time can be effectively reduced in great scale.

Basically, *FRAMS* is designed for general purposes. Even current prototype is more dedicated in setting up models for computer systems, it is to be expanded to model communication networks, manufacturing systems and other application areas. In the ensuing sections, the way of converting a model (i.e. an object) to a frame, the data structure of a simulation model in *FRAMS*, the operation of the data structure in simulation, the *FRAMS* prototype, and a study case are presented.

2 Models in Frame Configurations

By analyzing the architecture of an application system, it is naturally to decompose the system from top to bottom, i.e., a system is composed of several subsystems. In turn, a subsystem may be partitioned into some sub-subsystems. To the bottom, the elementary components are identified. Based on the layer-by-layer analysis, the architecture can be viewed as a multilayer-multicomponent configuration. Modern advanced systems usually have the feature such that the corresponding simulation models are able to be created systematically. By following the philosophy, the models used to construct a complete system model (which emulates a real system being investigated) will be organized hierarchically. The hierarchy facilitates the management of the models and the scheduling control in simulation. Meanwhile, the object-oriented programming techniques provides the features of information hiding, data encapsulation and the method of message passing. Through the involvement of these characteristics into a modeling and simulation environment, the system modeling process can be easily and systematically managed. But there is a problem which may occur at the model simulation stage. This problem can be profiled from Figure 1 in which the messages are passed up and down frequently in the hierarchical simulation structure. The path of message passing turns to longer as the complexity of a system model increases. The time spent to pass the messages directly affects the simulation time, especially in the experiments for the purpose of performance measures.

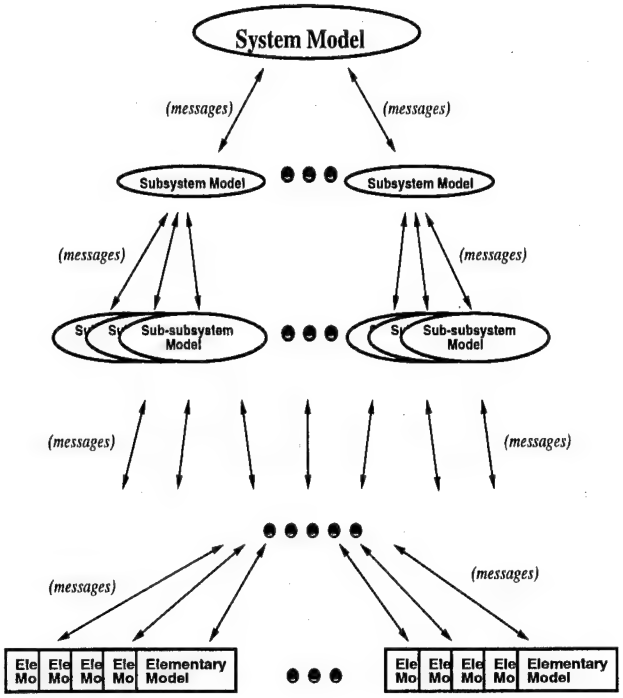


Figure 1: Message Passing in A Hierarchical Design.

By utilizing the object-oriented concept but ignoring the message-passing method, the frame configuration introduced in Artificial Intelligence [9] is employed to describe an object, i.e., a model. A frame is composed of slots including procedures (or called functions). The slot structure deserves the advantage of maintaining all the information related to the attributes of a model and the procedures required to figure out the performance data expected effectively and clearly. Furthermore, the hierarchical configuration of an application system can be effectively set up through the establishment of the relationship between frames. The slots defined in the *FRAMS*'s frame are listed below:

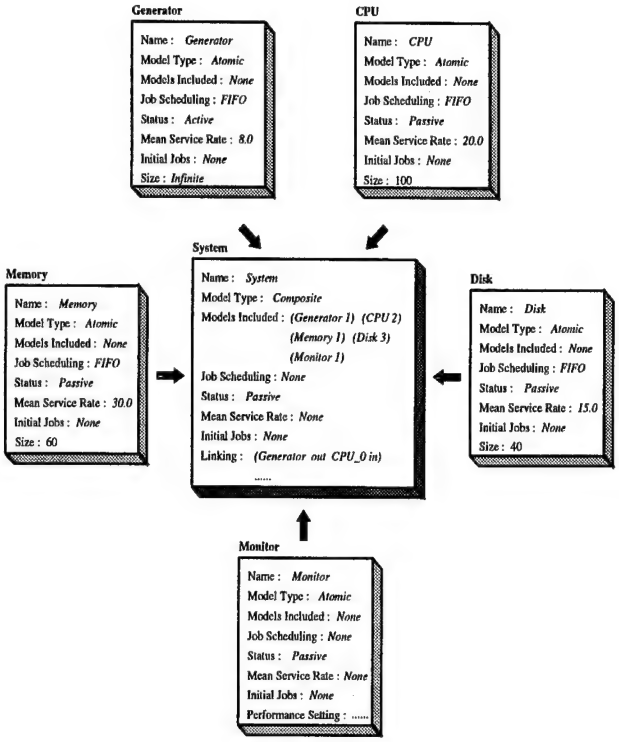
- *Name:* it records the frame name, i.e., the name of a model.
- *Model Type:* a frame can be either "composite" or "atomic" type.
- *Models Included:* the slot keeps the frames included and the corresponding instances required for a composite frame.

- *Job Scheduling:* the information handles the job processing sequence in a model.
- *Status:* it expresses the initial situation of a model in simulation.
- *Mean Service Rate:* this is the service rate of the model.
- *Initial Jobs:* the jobs set at the beginning of a simulation is maintained in the slot.
- *Size:* it is the buffer length in a model.
- *Performance Setting:* this slot holds the procedure(s) used for performance calculation.
- *Linking:* the coupling information between models are saved in the slot.

Referring to the example in Figure 2, the *System* frame is a composite frame in which the *Models Included* and *Linking* slots tells all the atomic frames involved and the related connections needed. The information about the atomic frames CPU, Memory, Disk, Generator and Monitor are depicted in the figure. This frame structure simply specifies the two-layer system architecture. By mapping an atomic frame to an elementary model, and a composite frame to other model in Figure 1, more complex application systems with multilayer and multicomponent organization is able to be systematically constructed with the proposed method. All the atomic and composite frames are kept in *Device Frame Base* (for short, DFB) and *System Frame Base* (for short, SFB), respectively. Both DFB and SFB associated with *Performance Frame Base* (for short, PFB) which is for computing performance data will be retrieved to organize the simulation models required at the *Device Modeling* and *System Modeling* stages explained in the next section.

3 The Data Structure of A Simulation Model

As mentioned early, there is no message passing occurred in the design of *FRAMS*. All the simulation activities are done by handling a hierarchical, multilink data structure built in a single process such that the simulation time spent compared to the same conditions but in message-passing simulation environment could be much smaller. This section discusses what the data structure looks like and how it is to be manipulated.



Generator
Name : *Generator*
Model Type : *Atomic*
Models Included : *None*
Job Scheduling : *FIFO*
Status : *Active*
Mean Service Rate : *8.0*
Initial Jobs : *None*
Size : *Infinite*

CPU
Name : *CPU*
Model Type : *Atomic*
Models Included : *None*
Job Scheduling : *FIFO*
Status : *Passive*
Mean Service Rate : *20.0*
Initial Jobs : *None*
Size : 100

Memory
Name : *Memory*
Model Type : *Atomic*
Models Included : *None*
Job Scheduling : *FIFO*
Status : *Passive*
Mean Service Rate : *30.0*
Initial Jobs : *None*
Size : 60

System
Name : *System*
Model Type : *Composite*
Models Included : *(Generator 1) (CPU 2) (Memory 1) (Disk 3) (Monitor 1)*
Job Scheduling : *None*
Status : *Passive*
Mean Service Rate : *None*
Initial Jobs : *None*
Linking : *(Generator out CPU_0 in)*
......

Disk
Name : *Disk*
Model Type : *Atomic*
Models Included : *None*
Job Scheduling : *FIFO*
Status : *Passive*
Mean Service Rate : *15.0*
Initial Jobs : *None*
Size : 40

Monitor
Name : *Monitor*
Model Type : *Atomic*
Models Included : *None*
Job Scheduling : *None*
Status : *Passive*
Mean Service Rate : *None*
Initial Jobs : *None*
Performance Setting : ......

Figure 2: A System Description in Frames.

3.1 Organization of The Data Structure

The organization of the data structure is illustrated in Figure 3. The multilink organization is composed of four kinds of nodes: A, B, C and D types as shown. Type A keeps the fundamental queueing properties of a composite model or an atomic model. Types B and D maintain the timing control and simple job in-out records. The information about model coupling is saved into type C's fields. The methodology applied to achieve the setup is described with two stages as follows:

Stage 1: Device Modeling The task is necessary whenever a new elementary device is modelled. The steps for the modeling are:

1) Assign the name to a device (the name of an atomic frame).
2) Define the related queueing properties for the device.
3) Create the atomic frame expected like in Figure 2, and save it to DFB.

Stage 2: System Modeling It consists of two parts: first to set up a composite frame, and then

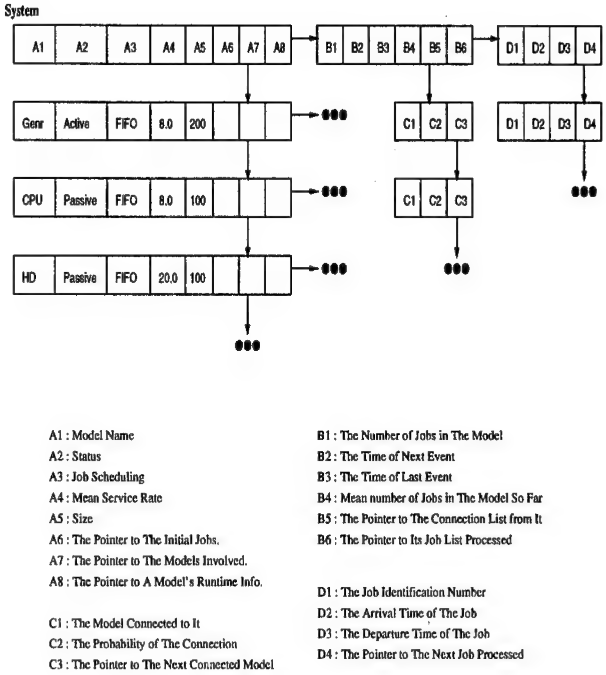

System
A1
A2
A3
A4
A5
A6
A7
A8
B1
B2
B3
B4
B5
B6
D1
D2
D3
D4
Genr
Active
FIFO
8.0
200
C1
C2
C3
CPU
Passive
FIFO
8.0
100
HD
Passive
FIFO
20.0
100
A1 : Model Name
A2 : Status
A3 : Job Scheduling
A4 : Mean Service Rate
A5 : Size
A6 : The Pointer to The Initial Jobs.
A7 : The Pointer to The Models Involved.
A8 : The Pointer to A Model's Runtime Info.
C1 : The Model Connected to It
C2 : The Probability of The Connection
C3 : The Pointer to The Next Connected Model
B1 : The Number of Jobs in The Model
B2 : The Time of Next Event
B3 : The Time of Last Event
B4 : Mean number of Jobs in The Model So Far
B5 : The Pointer to The Connection List from It
B6 : The Pointer to Its Job List Processed
D1 : The Job Identification Number
D2 : The Arrival Time of The Job
D3 : The Departure Time of The Job
D4 : The Pointer to The Next Job Processed

Figure 3: The Data Structure of A Simulation Model in FRAMS.

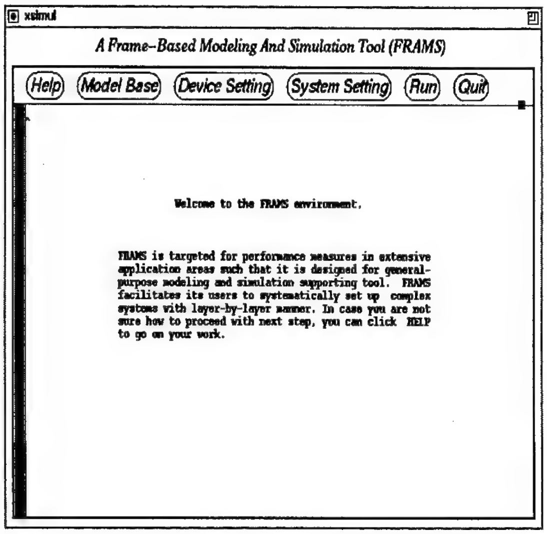
to generate the corresponding data structure for simulation purpose. The primary work includes:

1) Name the system (or a subsystem), i.e., the name of a composite frame.
2) Specify the required frames from DFB and/or SFB (which implies the multilayer design), and the number of instances for them.
3) Set up the connections (the couplings).
4) Construct type A's link list based on 1)'s and 2)'s settings but only *A1* field contains its name. Type *B* nodes are also created and pointed by *A8*.
5) Look up *A1*'s content to retrieve the atomic frame from DFB, and update *A2* to *A6*.
6) Based on 3)'s information to set up type *C* nodes and to feed the related data into *C1* to *C3*.

After the completion of applying the methodology, a data structure which represents the simulation model to an application system is organized. (The type D nodes will be created during the processing of the data structure.)

3.2 Manipulation of The Data Structure

For simplicity of explanation, the same configuration in Figure 2 is considered. The manipulation on the data structure during model simulation is activated from the places where *A2* fields have the status of "active". The "active" models can keep on producing jobs (the system traffic) until the simulation terminates. Each time a job is produced, a type D node will be created and linked to the place it belongs. The job will be dispatched to a specific model in terms of type *C*'s information. In turn, when a job arrives at a model, it will be queued into its buffer and wait for service under the policy of job scheduling. Meanwhile, a type D node for the job is generated and attached to the D node link. As soon as a job is completely served, it will be again based on type *C*'s information to be passed to another model. This situation is repeated to the end of the simulation.



A Frame–Based Modeling And Simulation Tool (FRAMS)

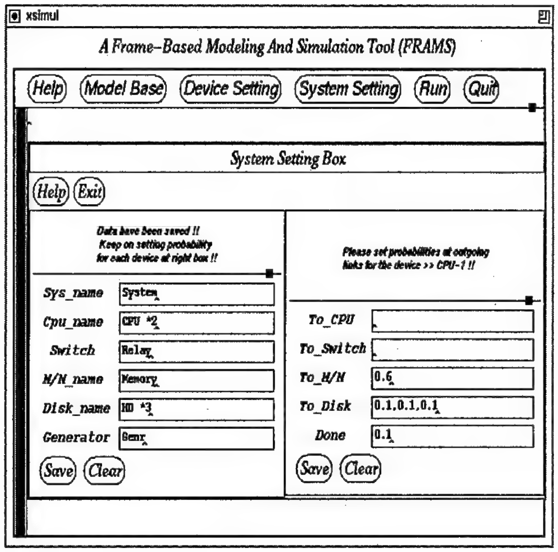
Welcome to the FRAMS environment.

FRAMS is targeted for performance measures in extensive application areas such that it is designed for general-purpose modeling and simulation supporting tool. FRAMS facilitates its users to systematically set up complex systems with layer-by-layer manner. In case you are not sure how to proceed with next step, you can click HELP to go on your work.

Figure 4: The Main Menu in FRAMS.

4 The Prototype of FRAMS

In order to enhance the functionality of *FRAMS*, a window-driven user interface has been developed. The interface not just facilitates users in modeling and simulation processes but encourages users to reuse the resources existing in frame bases. Figure 4 profiles the main menu with six window-click functions. The *Help* function provides a good direction to guide users to handle *FRAMS*. The *Model Base* is used to list the content in the DFB, SFB and PFB. Both *Device Setting* and *System Setting* functions form the modeling part. A successful operation at each *Device Setting* cy-



Data have been saved !!
Keep on setting probability
for each device at right box !!

Sys_name: System
Cpu_name: CPU *2
Switch: Relay
M/M_name: Memory
Disk_name: HD *3
Generator: Genr

Please set probabilities at outgoing
links for the device >> CPU-1 !!

To_CPU:
To_Switch:
To_M/M: 0.6
To_Disk: 0.1,0.1,0.1
Done: 0.1

Figure 5: The System Modeling Menu in FRAMS.

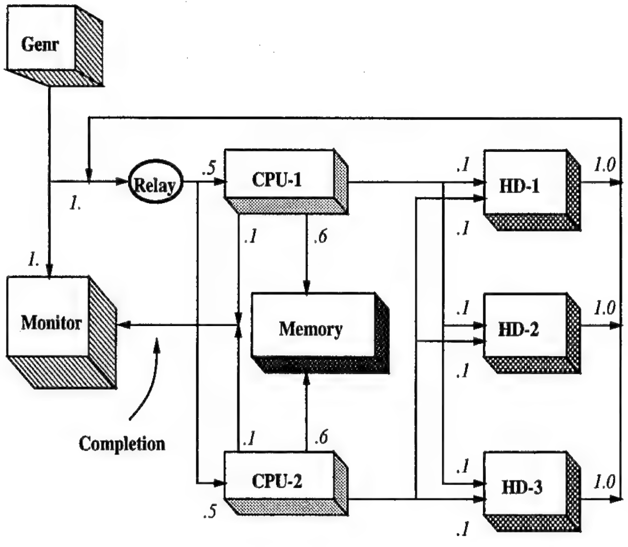


Figure 6: The Open-Queueing-Network System.

cle will produce a new atomic frame. Similarly, a composite frame can be created after executing the *System Setting* part completely. Model simulation is executed by clicking *Run*. The *Quit* is for leaving the *FRAMS* environment. In Figure 5,the detailed *System Setting* pop-up window is demonstrated. The *Help* function in the window teaches users how to do system modeling. The data in the left boxes show that the *System* model contains two CPUs, one Memory, three HDs and one Relay in cooperation with a job generator *Genr*. The probability settings for the outgoing jobs are specified in the right boxes. In the figure, only *CPU-1*'s settings are displayed. Other functions in the main menu are also driven by the similar pop-up window, and guided by the proper *Help* functions.

5 Analysis of The Experiments

As the design motivation in the research is based on Queueing Theory, the system models constructed in *FRAMS* can be classified as queueing networks. In [2, 4, 5], three fundamental types of queueing networks cascaded, open and closed queueing networks have shown their importance in extensive application areas. In order to verify *FRAMS*, the property of Burke's theorem is tested first. By comparing the *FRAMS*'s simulation outcome to the data calculated out by *analytical approach*, the high accuracy is achieved. Next, in Figure 6, an open queueing network having the following conditions is used to prove Jackson's theorem in *FRAMS*. Also, the scheme of *confidence interval* [7] is adopted to investigate the correctness of the simulation results. The conditions given to the devices in the figure are: 1) The mean service rates at CPU, Memory and HD (hard disk) are 50, 60 and 10 *jobs/millisecond*, respectively, 2) No jobs are blocked anywhere, 3) The values beside the lines express the job dispatching probabilities, 4) The generator Genr can generate jobs with 4 *jobs/millisecond*, and 5) Monitor is used to calculate average system time delay per job. The ideal average system time delay is 1 millisecond. In comparison with the ideal value, the outcome collected from multiple simulation runs shows that 95% confidence interval is carried out. This implies *FRAMS* is more reliable. The same result of testing Gordon-Newell networks is deserved as well.

The configuration in Figure 6 has been modelled by DEVS-Scheme [11] for comparison of simulation time spent. Even the actual time-saving scale is hard to be measured so far, the efficiency provided by *FRAMS* is overwhelming the former.

6 Summary

This paper presents a method of describing models with the frame configurations, proposes an approach to greatly shorten simulation time through the setup of the hierarchical, multilink data structure, and introduces the *FRAMS* prototype. However, further simulation time reduction is possible by employing the hybrid approach mentioned in [3, 10], and the scheme of multithreaded programming in cooperation with distributed computing. In order to achieve the improvement, the data structure used to form a simulation model should be adjusted like state-dependent queueing models allowed to be constructed. Both are critical to the future work. In addition, *FRAMS* will be enhanced to facilitate modeling in extensive application areas.

References

[1] *Reference Handbook*, CACI Products Company, 1993.

[2] P.J. Burke, The Output of A Queueing System, *Operations Research*, Vol. 4, pp. 699-704, 1956.

[3] K.M. Chandy, U. Herzog and L. Woo, Parametric Analysis of Queueing Networks, *IBW J. Res. Dev.*, vol. 19, pp. 36-42, 1975.

[4] W.J. Gordon and G.P. Newell, Closed Queueing Systems with Exponential Servers, *Opns. Res.*, Vol. 15, pp. 254-265, 1967.

[5] J.R. Jackson, Networks of Waiting Lines, *Operations Research*, Vol. 5, pp. 518-521, 1957.

[6] L. Kleinrock, *Queueing Systems*, Vol. I, John Wiley & Sons, Inc., 1975.

[7] A.M. Law and W.D. Kelton, *Simulation Modeling and Analysis*, 2nd Edition, McGraw-Hill, Inc., 1991.

[8] A.A. Pritsker, *Introduction to Simulation and SLAM-II*, Systems Publishing Corp., 1986.

[9] E. Rich, *Artificial Intelligence*, 2nd Ed. McGraw-Hill, Inc., 1993.

[10] T.G. Robertazzi, *Computer Networks and Systems: Queueing and Performance Evaluation*, Springer-Verlag New York, 1990.

[11] B.P. Zeigler, *Object-Oriented Simulation with Hierarchical, Modular Models - Intelligent Agents and Endomorphic Systems*, Academic Press, 1990.

AUTHOR BIOGRAPHIES

Lien-Pharn Chien is an Associate Professor in Department of Information Engineering at the Kaohsiung Polytechnic Institute, Taiwan. His research interests include design of modeling and simulation systems, fuzzy logic control, and applications in expert systems. He is a member of IEEE.

Ray Yueh-Min Huang is an Associate Professor in Department of Engineering Science at The National Cheng Kung University, Taiwan. His research interests are in system simulation, distributed computing environment and neural fuzzy network. He is a member of IEEE.

Hierarchical, concurrent state machines for behavior modeling and scenario control

Omar Ahmad
James Cremer
Joseph Kearney
Peter Willemsen
Computer Science Department
University of Iowa
Iowa City, IA 52242

Stuart Hansen
Department of Mathematics
University of Wisconsin — Stout
Menomonie, Wisconsin 54751

Abstract

This paper presents a framework for behavior modeling and scenario control based on hierarchical, concurrent state machines (HCSM). We present the structure and informal operational semantics of hierarchical, concurrent state machines. We describe the use of HCSM for scenario control in the Iowa Driving Simulator (IDS), a virtual environment for real-time driving simulation. The paper concludes with an outline of a forthcoming HCSM-based *scenario authoring* system that will permit non-specialists to graphically program behaviors and design experiments for IDS.

1 Introduction

State machines provide a natural framework for programming the behavior of synthetic entities in interactive simulation environments. The state machine methodology has been successfully used in a number of real-time control domains including robot walking and reactive systems[2, 3, 7]. Unfortunately, the absence of tools for abstraction and the lack of concurrency limits the usefulness of traditional state machines for programming complex behaviors[1, 8, 10, 11]. In this paper, we present a modeling framework based on *hierarchical, concurrent state machines* (HCSM) and demonstrate its usefulness for controlling entity behaviors and scenarios in real-time simulation environments.

Section 2 defines and provides informal semantics for HCSM. We present the structure of hierarchical, concurrent state machines and the state machine execution algorithm, explaining how output of concurrently executing state machines is resolved by higher-level machines. The execution algorithm is free of order dependencies that cause robustness and stability problems in behavior modeling. In Section 3, we describe how HCSMs can be applied to behavior modeling and scenario control for virtual environments. In particular, we describe the use of HCSMs to control vehicle behavior and to author scenarios for the Iowa Driving Simulation. In Section 4, we briefly describe a graphical programming environment for HCSM under development.

2 Hierarchical, concurrent state machines

State machines encode context dependent actions in a set of states. Traditional single-level, non-concurrent finite state machines can be used to model and control behavior by attaching output or *activity* functions to states. Activity functions implement a state's control law by computing control variable values appropriate to the state. At any instant, the single active state controls behavior by executing its activity function and returning control variable values. A state transition, in response to simulation events, yields a new active control law.

Traditional finite state machines treat all states with equal status and provide no means to organize groups of states into independent modules. The lack of encapsulation mechanisms inhibits reuse of state machine code. A group of logically related states can have transitions from any state in the group to any state outside the group; these transitions are left dangling when the group is surgically removed from the larger state machine. The inability to partition be-

0-8186-6440-1/94 $04.00 © 1994 IEEE

havior into separate modules complicates modification and extension of state machines; changes tend to propagate throughout the state machine.

The single-minded focus and sequential logic of non-concurrent state machines make it very difficult to satisfy the demands of problems that require simultaneous attention to many aspects of the environment. This encompasses much of the behavior we wish to capture in autonomous or intelligent agents. As the problem size grows, states proliferate to represent the response to factors occurring in various combinations and transitions tend to become dense and tangled. In the worst case, every existing state must be duplicated and connected to every other state to incorporate an intelligent response to a new factor in the environment.

To remedy these problems, we've extended the state machine model to include hierarchies of concurrent state machines. In our model, any state machine may contain multiple, concurrently executing substate machines. We find that the HCSM programs are easier to program, modify, and debug than the corresponding single-level state machine.

2.1 Hierarchical Concurrent State Machines Viewed as Black Boxes

From the outside, an HCSM state machine can be treated as a black box with input wires, output wires, and a control panel that contains buttons and dials. Information flows into the state machine over the input wires and values flow out of the machine over the output wires. The state machine is integrated into the simulation environment by connecting input wires to constants or variables and output wires to variables. More often, the input wires can be bound to expressions containing constants or simulation variables.

For example, Figure 1 shows the outermost view of a state machine for modeling driver behavior in vehicle control. Input wires provide values for driver aggressiveness and reaction time. The state machine outputs an acceleration and heading that are bound to the acceleration and heading of the vehicle. These values are passed to the dynamics subsystem and used to determine the position, orientation, and velocity of the vehicle. The use of input and output parameters and bindings allows us to design abstract state machines that are independent of the specific context in which they are used.

Control panels provide a means for state machines to communicate with one another during a simulation. A state machine can send messages to other state machines; a message can "push a button" on the control panel or adjust a dial on the control panel to one of its legal settings. For example, we may want a vehicle to turn right at the next intersection in order to set the stage for a scenario event. This can be requested by pushing the "turn right" button on the the vehicle's control panel. Buttons and dials permit state machines to interact with other state machines in ways that cannot be fully anticipated before the simulations begins. This ability is crucial to the orchestration of behaviors to create critical events and circumstances. For example, the traffic light state machine has a button that causes it to change from red to green.

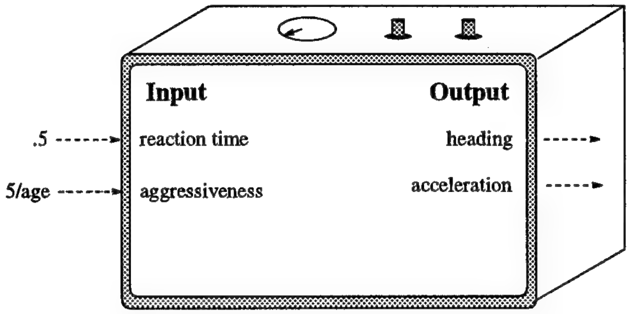


Figure 1: Blackbox view of an HCSM state machine.

The only difference between a button and a dial is in the persistence of their settings. A button is reset to its default position after it has been processed by the receiving state machine. A dial maintains its value until it is reset to another value. This distinction between transient interactions and influences that persist for a period of time is useful for behavior control.

2.2 The Structure of a Hierarchical Concurrent State Machine

In this section we present the internal structure of our hierarchical, concurrent state machines. An HCSM state machine is a structure of the form

$$< M, T, F, < I, O >, B, V >$$

where

- M is a set of state machines (generally referred to as *sub*-state machines)
- T is a set of transitions
- F is an activity function
- $< I, O >$ a set of input and output parameters
- B is a set buttons and associated button resolvers
- V is a set of local variables

2.2.1 States and transitions

The simplest state machine contains no sub-state machines and no transitions. Its activity function computes output values based on input values, local variables, and button and dial settings.

More complex state machines contain sub-state machines and may be categorized as either sequential or concurrent. In a concurrent state machine, there are no transitions — all the sub-state machines are active concurrently whenever the parent state machine is active. In a sequential state machine, exactly one sub-state machine is active at any instant. Transitions transfer control from one sub-state machine to another based on predicates involving the inputs, local variables and button and dial settings. The distinction between sequential and concurrent HCSM state machine's corresponds very closely with the distinction between AND-state and OR-states in Harel's Statechart formalism[10].

2.2.2 Activity functions

A state machine's activity function is responsible for computing output values based on the values returned from sub-state machines, input parameters, dial and button settings, and local variables. The activity functions can also send messages to other state machines.

The activity function for a sequential state machine is often quite simple; output values are computed based on the output values of the single active sub-state machine. When the sub-state machine outputs correspond directly with the parent machine outputs, the activity function often directly passes those values through.

On the other hand, the activity function for a concurrent state machine must compute a set of output values based on the output values of multiple active sub-state machines. Infrequently, a concurrent state machine's outputs are just the disjoint union of the the sub-state machine outputs; in such cases the corresponding activity functions are often simple, as in the sequential state machine case.

The multiple active sub-state machines usually correspond to competing control goals. Each state machine provides its own opinion about the values to assign to the control variables. The activity function implements a resolution method for these competing goals — it computes a set of outputs for the state machine based on the outputs of the sub-state machines. For example, in our driving behavior state machines a "most conservative" activity function yields good behavior in many (though not all) cases. For example, the top-level driving state machine has linear acceleration as one of its outputs. It contains sub-state machines that output linear accelerations for cruising, following, passing, and intersection behavior. The driving state machine's activity function can implement a "most conservative" rule by simply returning the minimum of the acceleration outputs of the sub-state machines.

Figure 2 shows the two views of an example HCSM state machine. The view on the left side exhibits the machine's hierarchical and concurrent structure and its state transitions. The view on the right depicts the flow of data through the machine, including the resolution of competing outputs from sub-state machines.

2.2.3 Information flow, input, and output

A parent state machine specifies how information flows into its sub-state machines by binding sub-state machine input parameters to expressions of local variables, constants, and input parameters. A sub-state machine receives information only when it is active. Likewise, sub-state machines produce output values only when they are active; when inactive, the output wires are dead.

2.2.4 History

State machines may retain their local state between activations. Thus, when a state machine is reactivated after a dormant period, sub-state machine execution can be reinitialized by activating the start sub-state machine or execution can proceed from the most recently active sub-state machine. Local variables declared to be static retain their values between activations of the state machine. Other local variables are reinitialized to default values on reactivation.

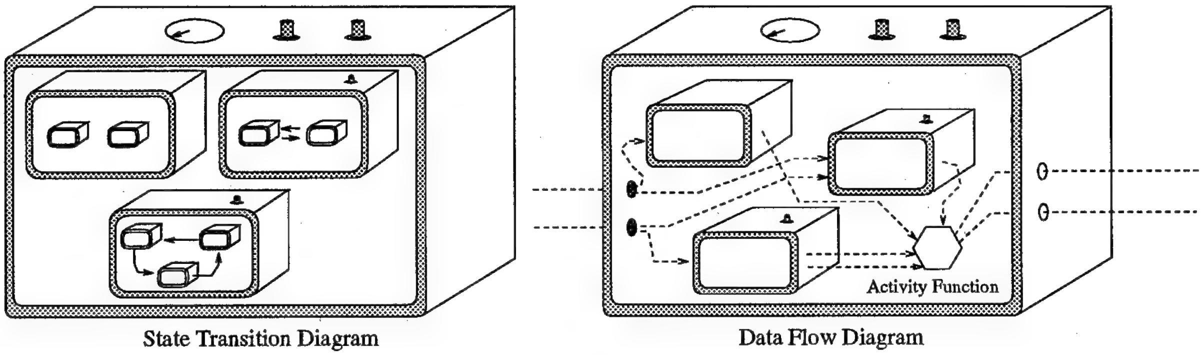
2.2.5 Button handling

The buttons and dials on a state machine's control panel may receive multiple messages simultaneously. Buttons and dials have *resolver* functions that are responsible for arbitrating between competing messages and ultimately determining button or dial setting.

2.3 Executing HCSM State Machines

The HCSM state machine execution algorithm is show in Figure 3.

The first step on each iteration is to resolve requests to set buttons and dials. The resolver function consid-



State Transition Diagram

Data Flow Diagram

Figure 2: Two views of concurrent state machine containing three sub-state machines. Two of the sub-state machines are sequential; the other is a concurrent state machine.

```
ExecuteHCSM (SM) {

   ResolveButtonsAndDials(SM);
   ComputeAndExecuteTransitions(SM);
   for all sub-state machines SSM of SM do
      ExecuteHCSM(SSM);
   RunActivityFunction(SM);
}
```

Figure 3: Basic HCSM execution algorithm

ers all requests in button and dial queues to determine settings.

The second step is to update the set of active state machines. Transitions from all active state machines are tested to see if their predicates are satisfied. Whenever a satisfied predicate is found, the predicate's source state machine is deactivated and the destination state machine is activated.

Next, sub-state machines are executed. The value of the overall state machine computation is independent of the execution order of the sub-state machines. Finally, the activity function is executed; it must produce state machine output values as a function of the newly computed sub-state machine outputs, button and dial settings, and local variable values.

3 Behavior modeling and scenario control using hierarchical, concurrent state machines

We developed HCSM to support behavior modeling and scenario control in virtual environments. In earlier work, we developed a hierarchical, concurrent state machine framework, the Conceptual Control Modeler (CCM), for specifying behaviors of high-degree-of-freedom mechanisms in rigid-body dynamics simulation [8, 9]. CCM provides control programming tools that are useful for developing animations and simulations of human, robot, and insect walking, robotic hand manipulation strategies, and interacting robots (e.g. robots assembling something or playing games). CCM was developed for use with the Newton system [6] and similar rigid-body dynamics simulators.

Our work on HCSM has been strongly influenced by other research on hierarchical, concurrent state machines and control methodologies for satisfying of multiple, concurrent goals. Harel's Statechart formalism [10, 11] elegantly extends state machines to include hierarchy and concurrency. Reynold's [12] presents a method to manage competing goals in his work on flocking behaviors. Brooks uses hierarchies of state machines organized by levels of competence to program robot walking. [2, 3] Lower level state machines implement primitive behaviors. Higher levels subsume levels by inhibiting or suppressing data paths.

Our work is strongly motivated by the needs of the Iowa Driving Simulator (IDS). The IDS is a high-fidelity operator-in-the-loop ground vehicle simulator that incorporates a motion platform, force feedback and control loading, high-quality visuals, sound, state-of-the-art real-time multibody dynamics, and scenario control. A Ford Taurus cab is mounted inside a dome[1] on top of the motion platform. High resolution, textured graphics are projected on screens on the dome walls — the forward field of view is 191° × 45° and the rear field of view is 64° × 35°.

The objective of our work is to create a methodol-

[1] Other cabs can be installed in the dome — IDS uses HMMWV and Saturn cabs for some of its projects.

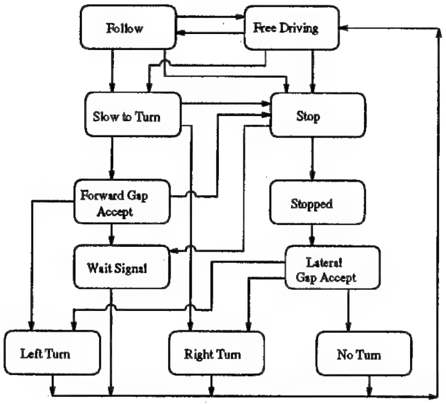


Figure 4: Simplified version of first generation IDS vehicle behavior state machine

ogy for controlling scenarios in IDS. The scenario control subsystem is responsible for generating and managing traffic, regulating traffic control devices, and setting the lighting and weather conditions. We partition the scenario control problem into two parts: basic behavior modeling and scenario authoring.

3.1 Basic behavior modeling

In the IDS, object behaviors are controlled by autonomous state machines that react to each other as well as to the motion of the operator's vehicle. A vehicle's state machine is responsible for controlling the position of the vehicle at each step in the simulation. A road database maintains information about the state of the virtual environment that is used by the state machines to fire state transitions and to set parameters in control laws that determine vehicle motions. All state machines are executed in sequence on a single CPU at 60Hz. At the end of each computation cycle, the road database is updated and vehicle locations are reported to the visual subsystem for display.

In the first implementation of the IDS, scenario control vehicle behavior was modeled using complex, one-level state machines. These state machines modeled driving on an open road, following behind another vehicle, and intersection behavior. Figure 4 shows a simplified version of a state machine used to model typical driving behavior. The actual state machines used in IDS are significantly more complex and includes states for passing and merging behaviors. Using these state machines, the scenario control subsystem can generate traffic that has a natural, reactive feel, and in which phenomena such as jams and clustering emerge. However, as mentioned in the previous section and detailed in [4], the model is difficult to modify and debug.

The second generation of scenario control uses HCSM to model object behaviors [5]. The ability to organize state machines hierarchically permits coherent activities to be grouped. For example, we have separate state machines for passing, following, and merging behaviors. The ability to encapsulate the logic of one aspect of behavior, such as passing, in a single state machine simplifies the development, modification, and debugging of control programs.

The concurrency of our state machines facilitates the creation of control programs that must simultaneously attend to the many factors influencing driving behavior. Vehicles must obey speed limits, stop at red signal lights, avoid collisions with nearby traffic, and be alert to hazards in the roadway. At each instant, a driver must integrate all the relevant information and adjust the accelerator and steering wheel to best accommodate the demands of safe driving.

Figure 5 shows the graphic environment we developed to aid in testing and debugging HCSM-based vehicle behaviors. At the top level there are six concurrent state machines controlling the vehicle. At every simulation step, each state machine independently produces a recommended acceleration for the vehicle. The activity function in parent state machine must compute a resolved acceleration based on the recommendations of the sub-state machines.

3.2 Scenario authoring

The ultimate goal of scenario control is to create a convincing dynamic environment for participant drivers. In the previous section, we described the techniques used in IDS to model the basic behavior of vehicles and traffic control devices. Using these techniques, we can generate ambient traffic composed of autonomous reactive vehicles.

Many of the applications for which the driving simulator is most useful require that drivers be exposed to specific crash threats. Investigators studying driving safety want to expose subjects to critical situations such as cars encroaching into their lane, unexpected braking by the vehicle in front of the subject's vehicle, and cars illegally driving through red lights. These experiments require scenario vehicles to perform specific actions in predetermined situations. Moreover, in order to compare performance across subject groups, experimenters demand that circumstances be replicable from trial to trial. The challenge we face is to create repeatable events by choreographing the behaviors of objects without sacrificing the sense of spontaneity characteristic of natural driving environments. We

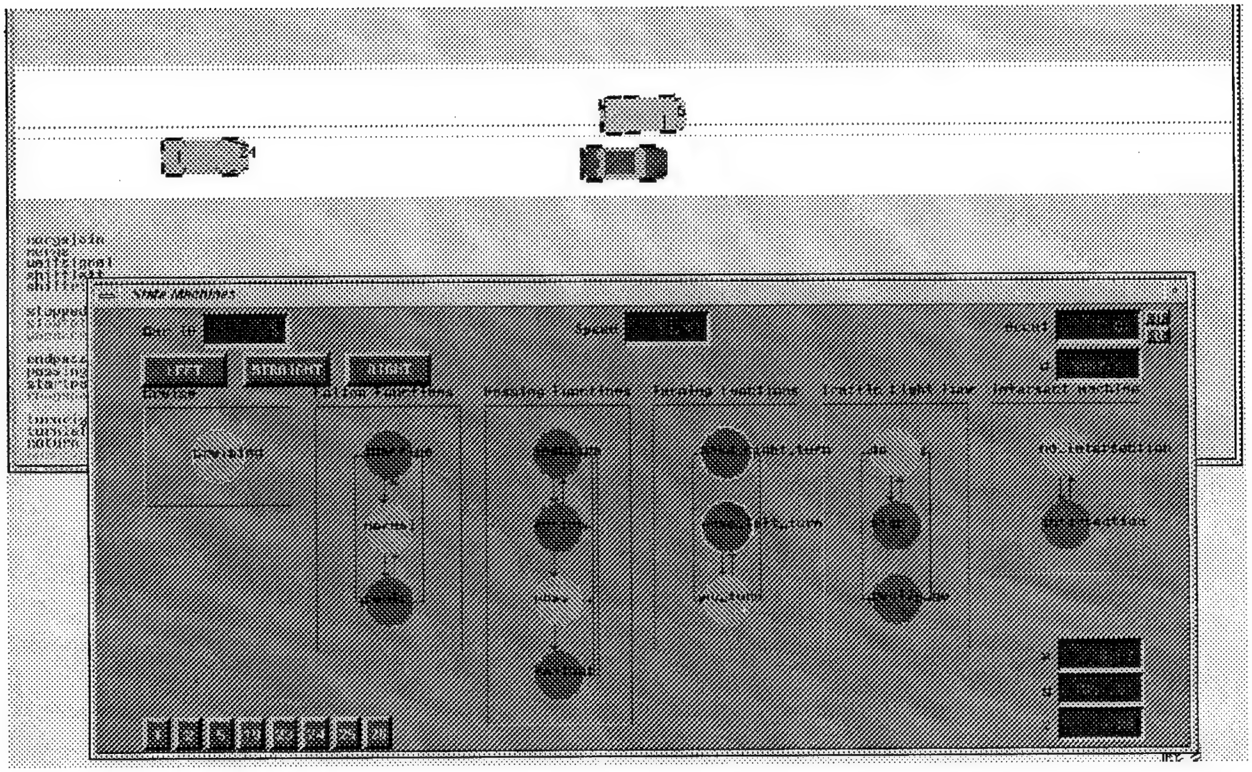


Figure 5: Prototype scenario editor and debugger.

want to inconspicuously direct the scenario so that the subject cannot anticipate upcoming events.

To accomplish this surreptitious control we have developed *behavior controllers* that manage situations by directing the behaviors of vehicles and traffic lights. Behavior controllers are HCSMs that behave like a daemon process in an operating system – they monitor situations and react accordingly. Behavior controllers influence the behavior of scenario objects by pressing the objects' buttons and setting dials.

We find it convenient to classify behavior controllers according to how they are activated and how they interact with other objects. For example, a trigger is a behavior controller that is placed at a specific location on the roadway and is activated when a vehicle drives over it. The trigger can be specialized so the it responds only to a specific vehicle or a specific set of vehicles. When the trigger is activated, it pushes a button on another object causing it to change its behavior. For example, a trigger can be used to initiate the motion of a vehicle on the shoulder of a highway as the subject's vehicle approaches it.

The behavior of a trigger is coded as a HCSM. This makes it simple to construct triggers that fire once or repeatedly. It is also simple to add delays between firings or create event sequences by chaining triggers so that one trigger activates another trigger.

A trigger is connected to a specific object. Sometimes we can't predict which particular objects must play roles in a creating a situation until the simulation is running. Inevitable differences in subject driving behavior lead to variations in the traffic that make it impossible to anticipate how a scenario will evolve. For these cases, we developed a *beacon* behavior controller. A beacon radiates messages to nearby vehicles. The beacon can be placed at a specific location or it can be attached to a vehicle. For example, a beacon can be attached to the subject vehicle and at the appropriate time instruct the vehicles in front to the subject to accelerate or change lane in order to create a clear path for the subject.

A beacon can be used to coordinate the actions of a number of objects. For example, consider an experiment in which we want to test a subject's response to a vehicle driving through a red light as the subject approaches an intersection. For a number reasons, it is undesirable to choose the vehicle to perform the offense off-line. Because subject drive at different rates they will arrive at the intersection at different times. Thus, it is difficult to guarantee that a particular car will be in position at the intersection at the appropriate time. Instead, a beacon can be used to watch for the subject vehicle and conscribe an appropriate scenario vehicle to run the light. The beacon may help set conditions for the event by sending "clear the way" messages to other scenario vehicles. In this way, behavior controllers can orchestrate complex scenario situations that retain significant reactive components and avoid the staleness of scripting.

4 Programming Environments for HCSM

At present, HCSM programs are coded in C. Programming state machines at this level is time consuming, tedious, and error prone. To aid program development, we are developing a high-level language for specifying HCSM state machines, a graphical editor for designing state machines, and a graphical inspector for visualizing state machine execution.

A prototype of the state machine visualization software is shown in Figure 5. The tool allows programmers to interactively inspect state machines as they execute on a graphic workstation. The tool has proven to be enormously useful for testing and debugging behavior models. Currently, work is under way to allow experimenters to define new behaviors by graphically creating state machines.

Acknowledgements

This research was supported in part by NHTSA Cooperative Agreement No. DTNH22-93-YU-07237 and by NSF grant CDA-9121985. Michael Booth designed and implemented the first generation of scenario control system for the IDS. Michael Klingbeil assisted in the development of HCSM, evaluating numerous variants of the basic framework.

References

[1] Jesse Aronson. The SIMCORE tactics representation and specification language. In *Proc. 4th Computer Generated Forces and Behavioral Representation Conference*, May 1994.

[2] Rodney A. Brooks. A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.

[3] Rodney A. Brooks. A robot that walks: Emergent behaviors from a carefully evolved network. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, pages 692–696, May 1989.

[4] James F. Cremer and Joseph K. Kearney. Scenario authoring for virtual environments. In *Proceedings of the IMAGE VII Conference*, pages 141–149, Tucson, AZ, June 1994.

[5] James F. Cremer, Joseph K. Kearney, Yiannis Papelis, and Richard Romano. The software architecture for scenario control in the Iowa Driving Simulator. In *Proc. 4th Computer Generated Forces and Behavioral Representation Conference*, Orlando, FL, May 1994.

[6] James F. Cremer and A. James Stewart. The architecture of Newton, a general-purpose dynamics simulator. In *Proceedings of the 1989 IEEE International Conference on Robotics and Automation*, pages 1806–1811, May 1989.

[7] Marc D. Donner. *Real-time Control of Walking.* Progess in Computer Science Series. Birkhauser, December 1986.

[8] Stuart A. Hansen. *Conceptual Control Programming for Physical System Simulation.* PhD thesis, Computer Science Department, University of Iowa, May 1993.

[9] Stuart A. Hansen, Joseph Kearney, and James Cremer. Motion control through communicating, hierarchical state machines. In *Proceedings of the 5th Eurographics Workshop on Animation and Simulation*, Oslo, September 1994.

[10] David Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[11] David Harel, Hagi Lachover, Amnon Naamad, Amir Pnueli, Michl Politi, Rivi Sherman, Aharon Shtull-trauring, and Mark Trakhtenbrot. Statemate: A working environment for development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.

[12] Craig W. Reynolds. Flocks, herds, and schools: A distributed behavioral model. In *Computer Graphics (SIGGRAPH 87)*, pages 25–34. ACM, July 1987.

Fluids in a Distributed Interactive Simulation

Chen Jinxiong and Michelle Sartor

Institute for Simulation and Training, University of Central Florida

Abstract

Today's training simulators have dealt mainly with vehicle dynamics, artillery dynamics and soil manipulations [9]. Important features such as fluid surface effects and flow over a terrain surface have been neglected, decreasing the realism of the simulation. The modeling and animation of fluids have recently been pursued vigorously in computer graphics, but fluid in a real-time networked virtual environment has not been studied. This paper investigates issues concerning the implementation of fluids in a Distributed Interactive Simulation (DIS). Several fluid models and a player/ghost simulation strategy are examined.

1.0: Introduction

In complex simulation and training systems, such as those supporting real-time interaction on a battlefield, a large number of simulation entities and a variety of geographical features are involved. A simulation training exercise can be more effective if subjects interact with their environment. Such environments may provide the capability to dig a ditch, build an embankment, leave tracks, generate dust, produce munition craters and crush vegetation. The addition of fluids to a simulated environment can increase its realism through affecting a vehicle's mobility as it fords a stream, allowing buoys and debris to drift with the current, flooding a landscape with water as a dam breaks, and generating a wake behind a boat.

This paper discusses a method for implementing fluids in a distributed interactive real-time simulation. Distributed Interactive Simulation (DIS) and the Dynamic Terrain (DT) project are briefly described to provide the reader with a background for this work. Three real-time fluid models of varying complexity are presented. Though other fluid models exist [5, 7, 12], the selected models were implemented due to their real-time performance capabilities. A player/ghost strategy for implementation in a real-time distributed interactive simulation is examined.

1.1: Distributed Interactive Simulation

Large virtual worlds in which many subjects interact with each other and their environment is an emerging capability of real-time simulation. The fruition of this interactive, simulated environment provides the tools for training of large-scale forces, rehearsal of missions, testing of new systems, and development of new tactics. All of these activities occur with no risk to life and allow replay of events.

This concept is the vision of the Advanced Distributed Simulation (ADS) movement sponsored by the Advanced Research Projects Agency (ARPA), Joint Warfighting Center (JWFC), Defense Modeling and Simulation Office (DMSO), and U.S. Army Simulation Training Instrumentation Command (STRICOM). Distributed Interactive Simulation evolved in an effort to make ADS a reality. The DIS community is working towards establishing a standards infrastructure that allows the interoperability of heterogeneous simulators on a distributed network [14].

0-8186-6440-1/94 $04.00 © 1994 IEEE

1.2: Dynamic Terrain project

The Institute for Simulation and Training (IST) was tasked by U.S. Army STRICOM to develop a real-time, malleable, simulated environment in order to train more effectively. Ground warfare involves extensive interaction with the terrain which can involve cratering from munitions, weather effects such as flooding, vehicle tread marks, and vehicle mobility. Initial work focused on real-time modifications to the terrain during a simulation. Extensions to this work include the terrain's effect on vehicle mobility and the addition of fluids to the simulated environment. DT researchers are exploring fundamental algorithms, data structures, and architectures that can support these complex models of a dynamic environment.

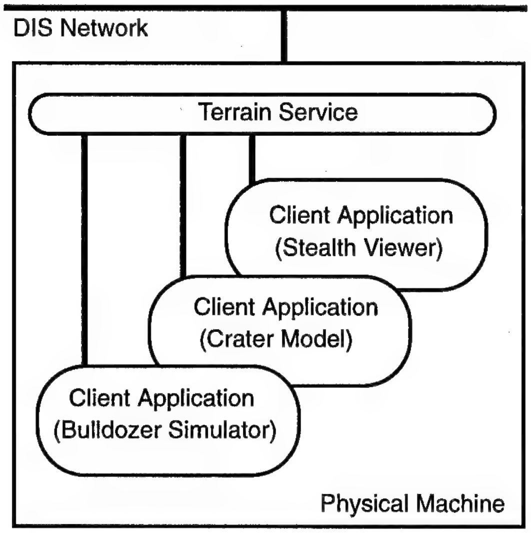

DIS Network
Terrain Service
Client Application (Stealth Viewer)
Client Application (Crater Model)
Client Application (Bulldozer Simulator)
Physical Machine

Figure 1: Sample DT service

The current DT system-level architecture evolved from a series of architectures which sought to allow unscripted entity/environment interactions in a networked simulation. This DT architecture is a client/server approach in which a service runs in the background, providing requested data to its client applications. Terrain Services is a service currently implemented in the DT simulation suite (Figure). Client applications such as a bulldozer simulator, a cratering model, and a stealth viewer receive modifications and send updates to an active terrain database during a simulation exercise via the Terrain Service. The DT architecture's flexibility allows for a central server, fully distributed, or hybrid configuration of the Terrain Services. The selected configuration is transparent to client applications - a benefit of encapsulation of the service mechanisms.

2.0: Fluid models for real-time simulation

The modeling and animation of fluids have captured the attention of the computer graphics community resulting in the development of many different fluid models. However, we are concerned with the trade-off between realistic and real-time fluid simulations which can be applied to a DIS environment.

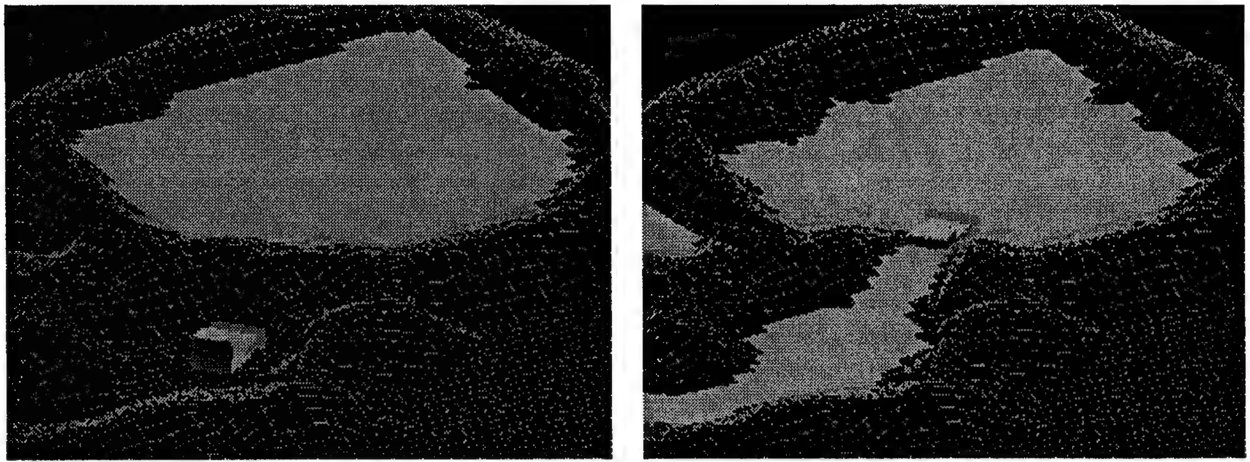
A variety of fluid models can be selected depending on the desired fidelity and available simulator computation power. A flat surface can be used to pictorially represent fluid, a low fidelity model allows limited fluid appearances, a high fidelity model achieves realistic fluid surface behaviors. All three fluid models have been simulated in real-time at IST using commonly available Silicon Graphics workstations, as powerful as or better than an Indigo. Fluid models which take significantly longer than real-time were not considered for DIS. The trade-off between fluid models is, with a low fidelity model, we have faster calculations but less realistic fluid behaviors, while with a high fidelity model, we have complex calculations but with more realistic fluid behaviors. It is noted that the models discussed in this paper make simplifications to accommodate the real-time demand. The approximations are not well-suited for a scenario which requires precise fluid dynamics, but they are sufficiently realistic for DIS applications. These models accommodate changing bounding edges through volume conservation. Fluid surface phenomena are achieved with the low and high fidelity models presented below.

2.1: Flat surface model

A flat surface representation is the simplest representation of a fluid. There is no surface behavior or fluid phenomena displayed. The perception of fluid is conveyed purely by its color. It is displayed as a flat surface of a particular height. Entities, such as boats, in the fluid do not generate any surface changes or fluctuations in boundaries. The fluid surface is merely lowered and raised on the surrounding terrain.

The advantage of this model is that it requires little computational power. A fluid surface is simply a polygon parallel to the X-Y plane. This trivial representation resembles the fluid's color at a desired location. This simple model allows a minimum fluid representation at a small cost to the simulator.

A simulator with little available CPU or without a need for complex fluid dynamics will find the flat surface model attractive. An aircraft simulator may only need fluid location for flight reference and does not have the additional computational power to expend on generating the fluid's surface behavior.



(a) before breaching dam

(b) after breaching dam

Figure 2: DT bulldozer/fluid simulation

2.2: A low fidelity model

Kass and Miller [8] presented a real-time method for animating fluid using a simplification of the shallow water equations. Their method handles wave refractions, wave reflections, the net transport of water and boundary changes with changing topology. This model allows such phenomena as flowing rivers, raindrops hitting the surface, and waves lapping (not breaking) on a beach.

Three approximations bring about their simplified form. First, the water surface is treated as a height field; thus, it cannot splash or break. The second approximation ignores the vertical velocity of a particle of water. This assumption causes the accuracy of the water model to degenerate if the waves become very steep. Finally, they treat the horizontal velocity of a column of water as approximately constant which breaks down as the water becomes turbulent. These simplifications approximate the classical linear wave equation.

Campbell [2] extended Kass' fluid model to accept floating bodies. This involved fluid displacement and the creation of ripples to simulate disturbances caused by floating objects. This model was integrated with a DT bulldozer. The dam can be broken by the bulldozer, resulting in a water spill (Figure 2).

The advantage of this model lies in its few calculations which are linearly proportional to the number of height-field samples. Though the fluid behaviors are limited and may not be quite realistic, animations occur in real-time. We consider this a low fidelity model because phenomena such as moving objects (boats) and surface ripple effects are too computationally expensive for a real-time simulation. This model exhibits the effects of a generic fluid, water, independent of its density and viscosity. The Reynold's number is an indication of these parameters which distinguishes fluids from one another. These capabilities are offered by the high-fidelity model.

2.3: A high fidelity model

The Navier-Stokes equations represent Newton's second law ($\Sigma F = ma$) in fluids and are the governing equations for general fluid flow [6]. Neither the flat surface nor the low fidelity implementation used these equations. The difficulties in a real-time implementation of these equations can be attributed to the effort involved in deriving a solution technique and in the run-time complexity of a solution.

To accomplish a real-time simulation, Jinxiong and Lobo [4] solved the two dimensional Navier-Stokes equations for the fluid's surface instead of calculating the fluid behavior for the full volume. The third dimension, fluid height, was obtained from the corresponding pressures in the flow field. When fluid from neighboring points flows into a single location, the pressure as well as the height of the fluid surface rise. When fluid exits a particular location by flowing to neighboring points, the pressure as well as the height of the fluid surface drop (Figure 3).

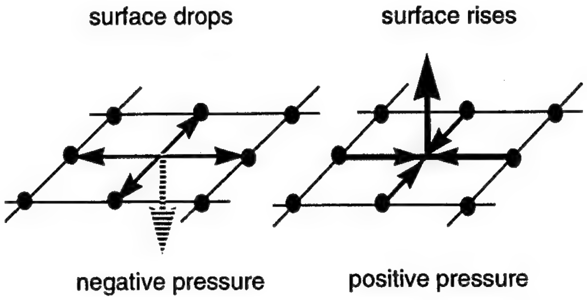


Figure 3: Fluid surface height

Session 1C:

Terrain Modeling and Reasoning

Terrain Modeling on High-Fidelity Ground Vehicle Simulators

Yiannis E. Papelis
Center for Computer Aided Design
The University of Iowa
Iowa City, Iowa

Abstract

Distributed Interactive Simulation (DIS) involves a large number of geographically dispersed operator–in–the–loop simulators interacting in the same virtual environment. Organized exercises in a DIS environment allow testing of group–level operations or procedures that require cooperation among large number of units. Recently, the ability of DIS is being expanded to evaluate equipment performance for the purpose of design modification and eventually, acquisition. This requires use of simulators that model the behavior of ground vehicles or other such equipment with enough detail to extract useful data from the simulator. Terrain modeling and representation becomes a central issue due to the significance of properly modeling the ground to vehicle interaction. This paper discusses design issues associated with modeling terrain for high–fidelity ground vehicle simulators. Experiences have been drawn from research leading to the design and implementation of an arbitrary resolution terrain model in the Iowa Driving Simulator (IDS), a high–fidelity ground vehicle simulator that is being integrated in the DIS network.

1: Introduction

All ground-based simulation models must have some knowledge of the terrain in which they operate [1]. Here, the term *terrain* refers to the physical geometry of the virtual environment and does not include logical attributes such as objects, roads or individual features. The degree to which the terrain needs to be modeled depends on the final purpose of the simulator. While for certain applications perfectly flat terrain at an arbitrary elevation may be sufficient, more often polygonal representations of terrain are more appropriate and provide more realism for a modest increase in complexity. Generally, the resolution of a polygon representation varies, but is often the same as the resolution of the Computer Image Generator (CIG) used in the system. In fact, the Height Above Terrain (HAT) function provided by most CIGs is a convenient method by which terrain information is negotiated in a system. As application domains become more demanding of simulators however, such approaches to the modeling of terrain become insufficient, and new methods must be developed for modeling arbitrarily complex terrain. One such domain is extending the use of the existing DIS framework and infrastructure to include the design evaluation, modification and eventually acquisition of ground–based equipment. Achievement of such a goal requires integration of high–fidelity, operator–in–the–loop ground–vehicle simulators into the DIS framework. This poses several problems stemming from the differing requirements that have driven the technology of simulators used in DIS versus high–fidelity simulators traditionally used for design evaluation and modification.

Evaluating the performance of alternative vehicle designs without using a physical prototype requires use of computer models with enough fidelity to provide engineering-level performance data in the course of a simulation. Such data can then be used for evaluation and potential modification of the design followed by re-evaluation in the same manner. This cycle, often referred to as virtual prototyping, may replace the traditional hardware–based design/test/modify iteration. Due to the tight integration between the human operator and device under design, achieving this virtual design cycle requires integration of the human operator in the simulation process. High–fidelity, operator–in–the– loop simulators achieve this goal by combining high–fidelity dynamic models with a rich set of feedback cues that create the illusion of operating the actual device. In such an environment, modeling of the interaction between the device and the operator is at least as important as modeling the device itself. Elements such as vibrations, large–scale motion cues, tactile feedback and sounds become an important part of the overall experience. In ground vehicles, the cause of such feedback is largely due to the interaction between the vehicle and the terrain. As a result, an accurate model of the terrain is a necessary component of a high–fidelity simulation. Resolution fine enough to capture the majority of the vehicle–terrain interaction, categorization of the property of the materials on the terrain, deterministic real-time interrogation, and support of arbitrary–complex databases which might include overlapping terrain are some of the properties that

0-8186-6440-1/94 $04.00 © 1994 IEEE

must be associated with a terrain model associated with high–fidelity ground–vehicle simulators. Due to the lack of requirements dictating such stringent fidelity in the terrain model, current DIS terrain implementations lack some or all of these properties. In typical SIMNET databases gridded terrain at Level 1 (90 meters), or Level 2 (30 meters), provides linear triangle patches that include information on the surface type. This information is sufficient for following the contour of the terrain and for calculating concealment. In situations where finer resolution is necessary, micro–terrain can be used [2]. Micro–terrain consists of a network of points that are joined to create a series of adjacent triangles. Information associated with each point includes elevation and soil type. Decoding the information embedded in micro terrain involves searching the network of triangular patches for one that includes the requested point and then interpolating to determine the elevation of arbitrary points within the patch. The computational cost of performing a terrain query depends heavily on the implementation, but generally is not of fixed–time complexity, a necessary feature for deterministic operation.

Successful integration of high–fidelity simulators in the DIS framework requires either that the existing facilities be extended to support the requirements of high–fidelity simulators, or that new approaches be used to model terrain. In the latter case, the compatibility among simulators participating in coordinated DIS exercises utilizing terrain models of varying capabilities must be addressed. This paper discusses some of the technical constraints associated with high–fidelity terrain models and presents a design that does not depend on facilities provided by SIMNET databases. The approach described here has, to some degree, addressed the majority of the issues associated with arbitrary resolution terrain modeling.

The remainder of the paper is organized as follows. Section 2 discusses in detail the requirements that must be addressed in the design of terrain databases in the scope of simulation and design evaluation. Section 3 describes the design of the terrain model used in the IDS [3], a high–fidelity operator–in–the–loop ground–vehicle simulator built as a testbed for simulator technology and virtual prototyping. Finally, section 4 discusses future directions.

2: Requirements on high–fidelity terrain models

2.1: Deterministic execution

One of the central requirements on an operator–in–the–loop simulator is hard real-time execution. Here, the term *real-time* refers to execution of one iteration of the simulation in less physical time than the simulated time. The term *hard* indicates the high degree of importance in maintaining real-time execution. The consequences of not maintaining real-time execution vary depending on the software component that failed and the overall design of the system. Generally, inability to maintain real-time execution results in events such as rendering delays or other cue discontinuities that are distracting to the subject, destroy the overall realism of the simulator, and can even be the cause of simulator sickness[4]. Clearly, the ability of the terrain model to provide deterministic queries is critical in minimizing such cue discontinuities. Because of the small integration steps typically utilized by multi-body dynamics, and because separate queries must be performed for each vehicle contact point, a terrain model is required to provide queries at a high rate. For example, a four-wheeled model that executes at 120Hz interrogates the terrain database at 480Hz. Performing more extreme maneuvers generally requires that the integration step be lowered. More complex vehicles also require more than four contact points to effectively simulate vehicle behavior. Such requirements eliminate the CIG HAT function as a plausible solution to the terrain interrogation design problem.

Another direct implication of the stringent deterministic execution requirement of a high–fidelity simulator is that techniques optimizing average performance cannot be used. For example, cacheing of disk data is often used in SIMNET terrain implementations as a means of reducing the average elevation lookup time. Such an approach makes no guarantee however, about the maximum execution time of a single elevation query. Scalability is another implication stemming from the requirement of deterministic execution. To ensure that the terrain model is scalable, any algorithm used in the process of resolving a terrain query must have a computational complexity that is constant (i.e., independent) to the size of the database. For example searches through the database would not be an acceptable solution unless there is a way to guarantee, before execution, a bound on the time it takes to perform the search. This bound must be fast enough to ensure real-time execution.

2.2: Resolution

The term resolution here refers to the density of information contained in the database itself. In general, whether or not the resolution in a database is adequate is directly related to the task that the simulator is used for. In virtual prototyping applications, where simulating and reproducing the maximum amount of cueing is important, the terrain database must support enough resolution to model all features that can cause cues detectable by the human operator. This includes elements such as general ground or roadway inclination, small hills, burms, potholes, and even the roughness of the surface. One design alternative, primarily used in DIS databases, is a coarse terrain model augmented by an object-based set of features that model individual elements. One of the features is micro terrain, a dense mesh of triangular patches that represents terrain in arbitrarily low resolution. The approach is targeted primarily for databases that consist of widely sparse areas with occasional areas of high–resolution terrain and is not well suited, even though it could be used, to modeling extended geographical areas with high–resolution requirements. Such areas include roadway with banked super–elevated turns, overpasses, trumpets etc., and closed test courses that intentionally contain extremely sharp curves or other features that push the stability and endurance of vehicles. The top view of Figure 1 illustrates a terrain scene of the former type as displayed by a CIG. The bottom view of Figure 1 illustrates the terrain elevation grid from a different viewpoint.

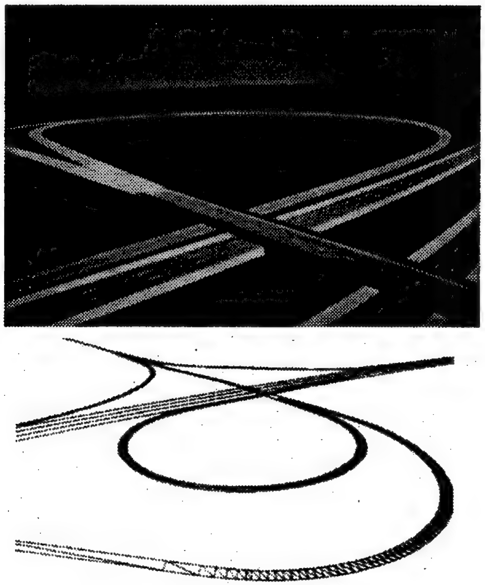


Figure 1: Terrain

Modeling of this type of terrain is important in virtual prototyping applications because it represents the ideal conditions under which to test newly designed vehicles. To capture the basic curvature of such terrain without detectable aliasing requires resolutions at or below 2 meters, locally with areas as low as .15 meters.

A solution that is often used for generating high–frequency cues without explicitly modeling the terrain below a threshold resolution is to use the terrain surface type as a selection criteria among a set of pre-defined set of recorded cues. For example, different background sounds, and different frequencies of vibration could be used when driving on concrete, asphalt, or grass. At runtime, the audio and motion simulator components could use the terrain type to select among the appropriate sound or vibration frequency. By not depending on the dynamics model to generate these high–frequency cues, this approach has many advantages. First, it is efficient, since it does not cause any runtime overhead and does not increase the complexity of the terrain modeling system. Furthermore, the simulator cues can be reproduced by playing back a recording of the actual cues which yield a realistic virtual environment and does not increase the modeling demands on the dynamics subsystem. Despite these advantages, this method cannot be used exclusively for terrain modeling, but it must augment an existing model.

2.3: Correlation with Other Databases

Due to the stringent execution time requirements imposed on the various simulator components, different representations of the same virtual environment are often used in a simulator. This allows customized views of the same virtual world that are optimized according to the specific needs of each cueing system. For example, the visual databases in almost all simulators consist of a polygonal representation that is specifically built to allow efficient rendering of the scene and contains a limited set of information regarding the logical contents of the scene. A separate object database may serve that purpose and depending on the application, a separate database could be used to provide terrain elevation. Representing the same virtual environment by using more than one representation allows room for correlation errors. A correlation error refers to the situation where the various representations differ. For example, it is not atypical to see polygons that have no other representation other than inside the CIG database, a fact evident when the simulated vehicle can penetrate them without any other cue from the remaining simulator components. To achieve realism and minimize the negative results of such distractions, the issue of correlation must be addressed in the terrain model design.

3: Terrain Implementation in the Iowa Driving Simulator

The IDS is a high-fidelity ground vehicle simulator serving as a testbed for the advancement of simulator technology, and is actively being used for a variety of purposes, including human sciences experimentation and virtual prototyping. The IDS is currently being integrated in the DIS network with interoperability planned for November, 1994. The IDS uses a multibody dynamic model that, depending on the complexity of the simulated vehicle and the specific application, executes at 120Hz, or 180Hz. Near–term plans include an increase in the execution rate of the dynamics to at least 240Hz.

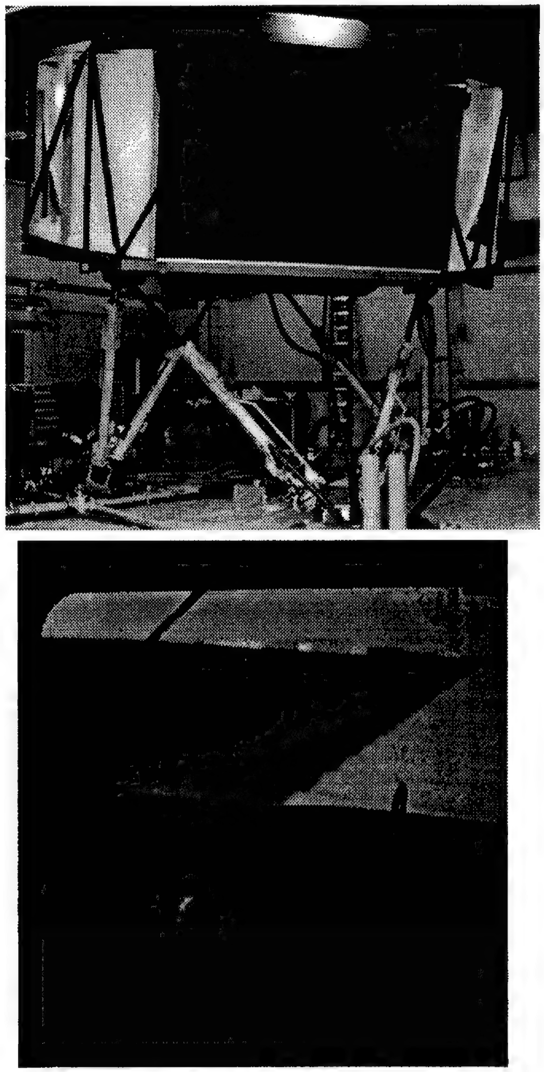


Figure 2: External and internal views of IDS.

Motion cues are generated by a high–payload hexapod motion base capable of a maximum of 1g accelerations at 8Hz. The top view of Figure 2 illustrates an external view of the motion base. Visual cues are generated by an Evans and Sutherland ESIG 2000 system that renders the scene at 50 or 60Hz, and audio cues are generated by a multichannel MIDI sound system. Tactile feedback is generated by a torque motor driven by the dynamics output that simulates the steering mechanism. All of these cueing subsystems are directly or indirectly affected by the tire-terrain interaction. Audio uses the surface type to modulate the tire rolling sound, while the tactile feedback is driven by a first principles simulation of the steering mechanism. The high bandwidth of the motion base allows it to reproduce high– frequency cues caused by the interaction of the terrain with the vehicle including low–frequency vibrations.

The virtual environments simulated by IDS can be grouped in two classes. The first includes databases populated with a variety of standard roadways which meet highway engineering standards and include banked superelevated curves. The other class includes models of actual test courses that consist of non-paved roads with sharp curves, extreme slopes and specialized areas used for vehicle stress–testing such as the Churchville Test Course and the Munson Test Area in Abberdeen, MD. To address these varying requirements, the IDS terrain database model supports variable density storage, provides an efficient and deterministic terrain query that is independent of the size of the database and supports overlapping terrain, a feature necessary for modeling bridges and overpasses.

The variable resolution storage is used to reduce the overall storage requirements and does not affect the efficiency of the terrain interrogation algorithm. The variable density storage is implemented by partitioning the ground (x–y) plane into *datazones.* A datazone is an arbitrary rectangle, aligned on the (x) and (y) axis, that contains *datasets*. A dataset includes information about the terrain at a specific location. This information includes at least the elevation and the type of terrain at the particular location. For storage and access efficiency, datasets stored to disk are organized in *buckets*, where each bucket is equal to one or a multiple of a disk block. Datasets are spaced at regularly distanced intervals within a datazone. The distance between adjacent datasets is defined as the *resolution* of a datazone. To resolve a database query, linear interpolation among the four datasets that surround the query point is used. Variable density storage is achieved by using multiple datazones for modeling a specific terrain database. Even though the resolution within a datazone remains constant, different datazones can have different resolutions. Use of an arbitrary number of datazones allows modeling terrain in different resolutions, based on the frequency content of different areas. Furthermore, datazones can overlap on the (x–y) plane allowing the modeling of vertically stacked terrain. An example of how several datazones can be partitioned is shown in Figure 3. The top figure illustrates a perspective view of the modeled terrain. The black dots indicate the endpoints of datazones, while the vertices of the rectuangular grid indicate location of the datasets. The bottom figure illustrates a top–down view of the datazone layout.

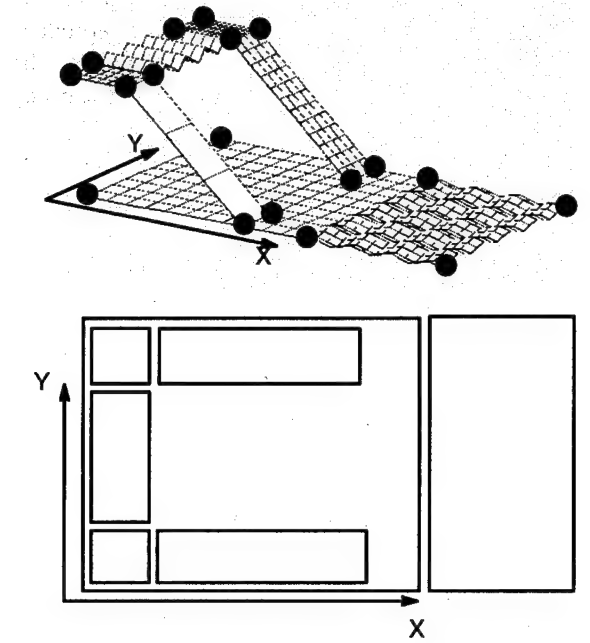


Figure 3: Example of variable resolution terrain representation in IDS.

3.1: Querying the Database

The input to a database query consists of the points (x), and (y), and an initial approximate elevation (z). The approximate elevation is required to resolve which datazone should be used if more than one datazone covers the point (x,y). For each datazone, the database stores the datazone rectangle boundaries, the datazone resolution, an overlap flag, and a pointer to the group of datasets associated with the datazone. The overlap flag is set when a datazone is stacked vertically with another datazone. The query algorithm proceeds as follows:

A Construct a list of datazones that cover the input X,Y point. For each of the datazones in the list:

A.1 Determine the four datasets that surround the input point

A.2 Retrieve the four datasets

A.3 Apply the interpolation formula and produce a query output

B Select the query output whose elevation is nearer the input elevation.

Constructing the list of datazones that cover the input point as required in step A, requires searching through the list of all datazones and applying a simple two–dimensional coverage test to each of them. To limit the time it takes to search through the potentially large number of datazones, the database uses a simple two–dimensional form of a hash table to partition the datazones based on their geographical location. The area of the database is divided into square regions called *sectors*. The sectors create a square grid that overlays on top of the terrain so that each datazone belongs to at least one sector. Associated with each sector is a list of datazones that belong to that sector. Since all sectors are square and have the same size, a few simple arithmetic operations can be used to determine the sector that the input point belongs to. Following this computation, the 2D coverage test need only be applied to the datazones that belong to that sector's list. Given that the sector size can be changed even after the database has been created, it is possible to subdivide a database to an arbitrary degree putting a bound on the maximum sector datazone lists and thus, putting a bound on the length of a search performed in step A. Figure 4 illustrates the partitioning of a set of datazones within sectors. Sectors are labeled with numbers while datazones are labeled with letters.

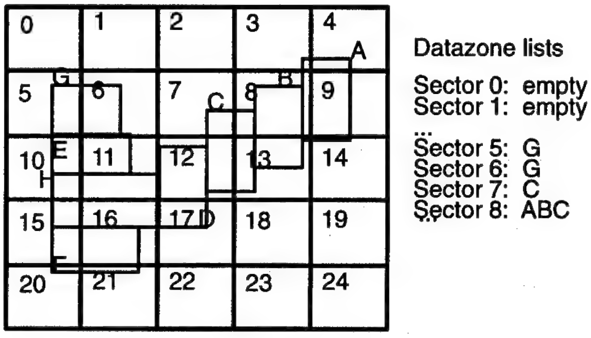


Figure 4: Sectoring the database

The remaining steps of the algorithm have fixed complexity that is independent of the size of the database. Step A.1 consists of simple indexing operations and has O(1) complexity to the size of the datazone. Retrieving the datasets as required in step A.2 is also a simple indexing operation, but has a potential for a large time penalty because the datasets are stored to disk. Step A.3 consists of a deterministic set of arithmetic operations and is of fixed complexity also.

Step A.2 requires accessing the datasets associated with a particular datazone in order to perform the interpolation. To increase disk access efficiency, datasets are packed together in memory blocks whose size is a multiple of the disk block (usually 2 or 4K). These memory blocks are called buckets. Making all buckets memory resident is not a viable solution given the large size of databases. To eliminate the disk access delays that would normally be associated with step A.2, a paging algorithm that predicts the location of the vehicle and reads buckets from disk before they are needed for actual queries is used. The group of buckets that resides in memory at any one time is defined as the *working set.* During initialization, the look-ahead algorithm reads enough data in the working set to cover an area that surrounds the simulator vehicle. At runtime, the algorithm maintains a bounding box around the driver and assigns relative priorities to buckets whose datasets are within the bounding box. These priorities are adjusted dynamically based on the vehicle's velocity and turning rate, as is the size, orientation, and position of the vehicle within the bounding box. For example, during high speed straight driving the bounding box is long and narrow, is aligned along the vehicle's path, and the buckets whose datasets lie immediately ahead of the vehicle are of the highest priority. An illustration of this configuration is shown in the top view of Figure 5. If the vehicle is traveling in slow speed and turning, the bounding box is shorter, wider and data on the turning side has higher priority. An illustration of this configuration is shown in the bottom view of Figure 5.

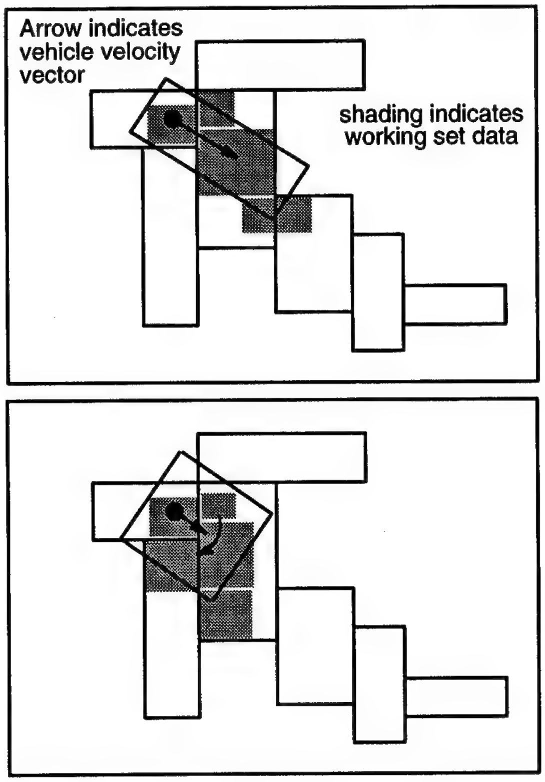

Arrow indicates vehicle velocity vector
shading indicates working set data

Figure 5: Operation of the Look-Ahead Algorithm.

The look-ahead algorithm is running as a regularly–scheduled process during the simulation. At each iteration, the algorithm computes which buckets are necessary, orders them according to priority, and compares them to the ones that are in the working set. If all buckets in the bounding box are in the working set, the algorithm doubles the size of the bounding box and then recomputes the buckets and their priorities. This process proceeds until the algorithms finds a bucket that is not in the working set and is of higher priority than the lowest priority bucket in the working set, or until the bounding box has been doubled a fixed number of times. In the former case, a disk read request is issued to fetch the new bucket into the working set. In the latter case the algorithm remains idle for one iteration.

Note that under certain assignments of datasets to buckets, it is possible to suffer internal fragmentation, where datasets stored within the same bucket cover a large, potentially non–contiguous geographical area. Such buckets may pose a problem because even though they occupy space the working set, only a small part of their contents contributes towards coverage of the bounding box. For example, note that in Figure 4 the shaded area, representing data in the working set, covers regions outside the bounding box. Internal fragmentation can be minimized by keeping the bucket size small, and ensuring that datasets stored in the same bucket cover geographical areas that are in close proximity.

The algorithm has few changeable parameters which include the initial size of the bounding box, the factors for scaling the bounding box according to the vehicle's movement, and the priority maps used to assign priorities to the buckets within the bounding box. Changing these parameters allow performance tuning in cases where internal bucket fragmentation reduces the effective coverage of the working set. Nevertheless, operation of this algorithm is based on the assumption that the performance bottleneck is disk access latency and not data throughput. The data throughput can be expressed as $\frac{W \cdot v}{res^2} \cdot DSS$ where W is the length of an imaginary sweeping line across which the vehicle must interrogate the terrain, v is the velocity of the vehicle, *res* is the terrain modeling resolution, and *DSS* is the size of a single dataset. Applying this formula to a vehicle that covers a 10 meter lateral area and is traveling at 100 mi/hr (44.7 m/sec) over terrain modeled at 0.1 meters with 8 bytes per dataset yields a data throughput of approximately 350Kb/sec. Such performance is not unreasonable, especially considering the extremity of the numbers used in the example. On the other hand, disk access latencies are routinely quoted at 10 to 15 msecs, and these numbers represent average performance figures and do not include any operating system overhead. The high execution rate of the simulator subsystems and the non-determinism of accessing a disk makes a read-on-demand approach infeasible. Use of the look-ahead algorithm decouples the non-determinism and latencies associated with disk storage devices from the terrain interrogation. Furthermore, since the lookahead algorithm does not need to be strictly synchronized with the vehicle dynamics component and only needs the long–term movement trend of the vehicle, it can run in parallel with the dynamics.

This approach to modeling and interrogating terrain has been implemented and exclusively used in the IDS since the beginning of its operation. A variety of databases have been modeled, some small enough to fit the working set while others being a orders of magnitude larger than the working set. The terrain interrogation has been measured at below 0.15 milliseconds on a 40Mhz i860 processor, and that measure is independent of the overall database size. On the same processor, the lookahead algorithm requires between 1 and 4 milliseconds, depending on the average number of buckets per datazone. Reading, a bucket can take between 5 and 200 milliseconds, and as a result the lookahead algorithm does not need to execute more often than 30 Hz.

4: Terrain database construction tools

To address the problem of correlating the terrain database with other databases used in the simulator, a set of software tools built within the Center is used to create all databases from the same specification [5]. This allows creation of visuals, terrain, and any other separate but correlated databases that are customized to the specific needs of various simulator components. Under an agreement with MultiGen, Inc. (previously known as Software Systems), a subset of these tools has been incorporated in MultiGen®. As part of this integration, the output capabilities of MultiGen® have been extended to include datazones in the format used by IDS. This integration allows effortless creation of datazones that are automatically correlated with a visual database. This method was recently used to create a variable–density terrain model for the Churchville Test Course and the Munson Test Area in Abberdeen, MD. The driveable part of these courses was modeled using datazones with resolution variances between 6, 2, and 0.15 meters yielding more than 3 million datasets for the Churchville test course.

Terrain databases can also be constructed by translating an existing terrain database into the internal IDS format. To ensure correlation with the visual model, the HAT function of the CIG can be used to sample the terrain at regular intervals off-line. This data can then be used to construct datazones. Alternatively, datazones can be constructed by directly translating the data of the original database. This method was used to create a terrain database for Fort Hunter-Liggett, CA, as part of integrating IDS in the DIS net.

5: Future directions

Tools need to be built that allow small modifications to the terrain database data following the initial creation of the database. Such modifications are necessary when the source data is represented in coarser resolutions such as DIS Level 1 or 2 databases. In such cases, it would be useful to have the ability to smooth the polygon-to-polygon edges and introduce roughness in the otherwise flat polygons. Smoothing would also be useful for reducing the feeling of "driving across a driveway," which is often encountered on the bases of hills or sides of mountains. Roughing a flat area would serve the same purpose as textures do in visual databases, which is to add content and roughness to an area that inherently is flat and boring.

Acknowledgement: Research supported by NSF–Army–NASA Industry/University Cooperative ResearchCenter for Simulation and Design Optimization of Mechanical Systems and ARPA.

6: References

[1] T. Stanzione, "Suitability of the Standard Simulator Database Interchange Format for Representation of Terrain for Computer Generated Forces,", In Proceedings of the Fourth Conference on CGF and Behavioral Representation, pp. 231–238, May, 1994.

[2] J. E. Smith, "Compact Terrain DataBase Library User Manual and Report," BBN Systems and Technologies, Bellevue, WA, 1991.

[3] J. Kuhl, D. Evans, Y. Papelis, R. Romano, and G. Watson Papelis, The Iowa Driving Simulator: An Immrsive Environment for Driving–Related Research and Development, manuscript submitted for publication, 1994.

[4] R.S. Kennedy, L.J. Hettinger, and M.G. Lilienthal, Simulator Sickness, Chapter 15 of Motion and Space Sickness, ed. G.H. Crampton (Boca Raton: CRC Press, Inc., 1990) pp. 317–341.

[5] D. Evans, "Correlated database generation for driving simulators," In *Proceedings of the IMAGE IV Conference*, pp. 353–361, July, 1992.

Terrain Reasoning Challenges in the CCTT Dynamic Environment

Charles E. Campbell and Gene McCulley
Science Applications International Corporation
Orlando, Florida

Abstract

The desire for realism in simulation is obvious. It follows that the more realistic the simulation environment, the more effective the training. A dynamic environment, in which man–made structures can be damaged, excavations are possible, and environmental conditions can change, is an aspect of simulated training attracting much attention. However, full implementation of a dynamic environment in a large scale simulation is not yet feasible. CCTT is taking the first step toward creating a dynamic environment in a production simulation system by allowing limited dynamic modifications to the static terrain database and environment. This paper details some of the challenges for terrain reasoning when implementing dynamic terrain, and explains the proposed approaches to solve them in CCTT's real–time networked environment.

1. Introduction

The Close Combat Tactical Trainer (CCTT) is the first trainer in the Combined Arms Tactical Trainer (CATT) family of training systems. CCTT is a real–time networked simulation environment designed to provide training of specific military skills at a fraction of the cost of an equivalent field exercise. CCTT is composed of several different types of systems including Manned Modules, Workstations, and Semi–Automated Forces (SAF). Manned Modules consist of crew cabin simulators, including M1A1s, M2A2s, HMMWVs, and dismounted infantry (DI). Workstations provides simulation capabilities for the battalion support staff, After Action Review, and simulation support. SAF provides additional friendly (BLUFOR) and enemy (OPFOR) entities by emulation of vehicle dynamics and crew behaviors. Each of these systems communicates using the Distributed Interactive Simulation (DIS) network protocol. An accurate representation of the battlefield terrain shared by all players is key to the interoperability of these distributed systems.

Increased realism in the networked virtual environment is essential for increased training effectiveness. The Institute for Simulation and Training has been researching the various components (e.g. algorithms, data structures, network protocols) needed to implement dynamic terrain for networked simulation [1]. Dynamic environment models have been implemented for a variety of areas, including excavation [2], cloud scenes [3], and water flow [4]. This level of dynamic terrain is computationally expensive, and is not yet practical for real–time networked simulation. As a result, distributed simulations have traditionally been implemented on static terrain databases, with almost no provisions for dynamic modifications of terrain components.

This paper provides a "snapshot" of current issues and future plans. Many issues are still being resolved. At the time of this writing, no CCTT databases with dynamic attributes have been generated. For more information on the CCTT SAF database design, see [5]. For more information on the CCTT visual implementation of the dynamic environment, see [6].

2. The dynamic environment

The following sections describe the various aspects of the dynamic environment to be implemented in CCTT visual and SAF databases.

2.1 Destructible static features

Destructible static features (DSF), also known as fixed selectable features, are terrain features which are placed in the static terrain database, but have normal, damaged, and destroyed states which may alter their geometry (Figure 1). All DSFs have a unique identifier to allow them to be referenced individually. DSFs include buildings (such as houses, industrial buildings, and water towers) and bridges.

0-8186-6440-1/94 $04.00 © 1994 IEEE

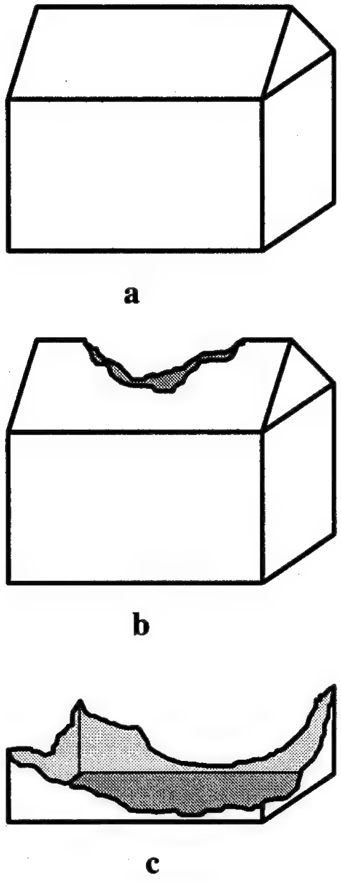


Figure 1. Destructible static feature example (a) normal, (b) damaged, and (c) destroyed

2.2 Dynamically placed features

Dynamically placed features (DPF), also known as relocatable objects, are terrain features which can be placed onto the terrain database during a simulation. DPFs may be surface obstacles, changes to terrain geometry, or other trafficability modifiers. DPFs can be damaged, destroyed, or breached. In CCTT, DPFs are placed by Combat Engineering entities to provide mobility, countermobility, and survivability.

A brief description of each DPF scheduled for the CCTT databases at this time follows. This list is preliminary, and items may be added, deleted, or modified.

A *log crib* is a rectangular obstacle made of stacked trees and used for countermobility. Log cribs can be damaged and destroyed. A destroyed log crib is no longer a trafficability obstacle.

An *abatis* is a countermobility obstacle consisting of felled trees aimed in the direction of the enemy. An abatis can be destroyed, which effectively clears the obstacle.

A *tank ditch* is a countermobility obstacle which alters the terrain skin. A tank ditch can be composed of up to four segments. Each segment can be 30, 60, 90, or 120 meters in length, and each has its own three dimensional orientation. A tank ditch can be breached. Breach locations are 15 meters from an end of the ditch, and every 30 meters thereafter.

A *concertina wire fence* is placed along a tank ditch to provide additional countermobility. A concertina wire fence has the same length, number of segments, and breach locations as the tank ditch it is associated with.

Fighting positions for DI are simply holes in the ground which are deep enough to hide a DI's body, yet allow him to engage the enemy. Fighting positions include *infantry fighting position*, *overhead covered infantry position*, *machine gun prepared position*, and *covered machine gun bunker*. The exact geometry of each model differs slightly, with berm placement variations. These positions can each be destroyed, rendering them useless.

A *hull defilade* is similar to an infantry fighting position, except that it is made to accommodate vehicles. Multiple types of hull defilades will be available to support the different types of vehicles (Armored Vehicle, Fighting Vehicle, Tank, Mortar Carrier). The hull defilade hides the vehicle hull while allowing the vehicle to engage the enemy. Hull defilades have an orientation, and thus must be entered from the appropriate direction. A hull defilade can be destroyed, rendering it useless.

A *minefield* defines a countermobility area in which explosives have been placed in the ground. Minefields can be breached to provide a trafficable lane cleared of mines.

An *Armored Vehicle Launched Bridge* (AVLB) becomes a DPF when launched from its vehicle. An AVLB is 60 feet in length and is used to cross gaps (rivers, ditches, etc.). An AVLB can also be used to breach concertina wire fences. An AVLB can be destroyed, rendering it useless.

Finally, two other DPFs have been discussed, *building rubble* and *ribbon bridges*, but very little about their implementation and use is known at the time of this writing, and so they will not be discussed here.

2.3 Natural environmental effects

Natural environmental effects describe states of naturally occurring phenomenon. In CCTT, these include rain, fog, haze, cloud cover, and time of day. All natural environmental effects in CCTT SAF will be represented as global, discrete states. For example, rain will be either on or off for the entire database, and the transition between on and off is immediate. In CCTT, natural environmental effects will affect mobility and visibility.

2.4 Man–made environmental effects

Man–made environmental effects describe states of environmental phenomenon introduced into the environment by a simulated entity. In CCTT, these are flares and tactical smoke. Man–made environmental effects will be used to restrict or enhance visibility.

3. The environment manager

A central environment manager will maintain all dynamic environment information. This eliminates the need for each simulator to maintain a copy of some environmental changes by maintaining a central repository of the information, saving space and time. The environment manager will communicate with other entities via DIS Protocol Data Units (PDU). The proposed Synthetic Environment Protocol is described in [7].

The environment manager does not yet keep track of natural environmental effects. This is because CCTT currently performs no weather modelling. Weather (and time of day) are defined as discrete states. If weather is modelled in the future, the environment manager will do the modelling.

Of the two man–made environmental effects, only tactical smoke is handled by the environment manager. An entity participating in the exercise tells the environment manager where to create the smoke. The environment manager then creates and models the smoke. Modelling smoke consists of changes in size and density. The environment manager broadcasts creation of and updates to the smoke to all applications in the exercise.

The environment manager monitors the network for munition impacts on or near DSFs. The environment manager then assesses damage to the DSF. If the damage is sufficient to cause a state change to the DSF, the environment manager will broadcast an update indicating the new state of that DSF to all applications in the exercise.

Creation and modification of DPFs is handled in a similar fashion. The creating/modifying entity tells the environment manager the necessary information to create and place the DPF. The environment manager stores the information and broadcasts the new DPF information to all applications in the exercise.

Finally, the environment manager maintains data for reset and checkpoint during an exercise. This provides a single source for environmental data at any checkpoint. Support for restarting/reconstituting entities is provided through a unicast send of the latest environment information. Unicast responses avoid interrupting every processor in the exercise.

4. CCTT SAF terrain reasoning

CCTT SAF has terrain reasoning requirements to provide height of terrain, collision detection, munition impact detection, obstacle avoidance, route planning, line of sight, and cover and concealment. Dynamic environments impose new challenges on terrain reasoning, because it forces SAF entities to be aware of changes to the environment and adapt functionality to account for the changes.

For clarity, descriptions for the terrain reasoning terms and functionality with respect to CCTT SAF are provided. DSFs and DPFs are referred to collectively as *dynamic features.*

4.1 Height of terrain

Height of terrain provides the elevation and surface type at a given location on the simulation database. This involves interpolation between three terrain elevation posts which constitute a terrain polygon. River beds will be modelled with a simple geometry, such that entities driving into a river will gradually immerse into the water, rather than suddenly dropping to the depth of the river.

To lessen the computational expense imposed upon terrain reasoning by the addition of dynamic terrain features, SAF entities will not drive or climb on any features placed on top of the terrain, with the exception of bridges. This aids computation for height of terrain by eliminating the need to calculate elevations on building rubble, abatis tree trunks, etc.

Height of terrain does, however, handle changes to the terrain skin such as tank ditches and defilades. Instead of using the terrain surface, height of terrain must detect that the query location is in fact on a DPF, place and orient the model, and interpolate the actual elevation.

CCTT SAF may add overpasses and bridges to the list of DSFs in the near future. Overpasses create a situation where one x,y location can have multiple valid elevations. At these locations, height of terrain must determine which elevation is desired. Height of terrain must recognize when a bridge or overpass span is destroyed.

Surface types vary widely throughout the CCTT terrain database. A simulated entity's mobility is altered by the type of surface it is traversing. Rain has the effect of saturating the soil, resulting in a different surface type with altered mobility characteristics.

4.2 Line of sight

Line of sight determines the percent visibility of an entity from a specified eyepoint. Percent visibility indicates the amount of the 2D projection of the target entity which is visible from the eyepoint. Visibility is degraded by solid obstructions, such as buildings or terrain skin, and by terrain features which provide partial transmittance, such as trees.

Line of sight uses entity and feature bounding volumes to determine the extent of blockage that entity or feature incurs. A *bounding volume* is a three–dimensional rectangular volume which defines the 3D boundary for an object, such as an entity or a terrain feature. Bounding volumes are useful for defining the extents of an object without using a complex geometrical representation.

Line of sight (LOS) becomes more complicated with the addition of dynamic objects. In particular, the irregular shapes resulting from object destruction present a LOS nightmare. CCTT has agreed that when objects are damaged, their geometries will not change drastically, and when objects are destroyed, their rubble will be roughly consistent in height across the object. This allows LOS to be simplified in that bounding volumes will not be affected in the damaged state, and destroyed objects (excluding bridges) will still use a rectangular bounding volume, although it will be lower. Obviously LOS calculations across damaged and destroyed objects will not be entirely accurate, but they will be close. More importantly, these simplifications allow LOS calculations to be performed efficiently without imposing unrealistic looking restrictions upon the visuals.

Line of sight is also affected by DPFs. The DPFs which sit on the surface (e.g. log crib) are simply another obstacle to be taken into consideration. Excavations, however, allow entities to be hidden partially or totally below the terrain surface. This is easily handled by LOS, since the terrain surface is already taken into account. The difficulty lies in the fact that excavations generally create a "berm" of soil around them. These berms are significant for fighting positions, in that they provide some degree of cover. Therefore, these berms cannot be ignored for line of sight.

Environmental effects can alter line of sight. Smoke, fog, haze, rain, and darkness degrade visibility. Cloud cover can completely block line of sight to high flying aircraft. Flares can increase visibility within the areas they illuminate.

4.3 Collision detection

Collision detection monitors simulated entities to determine if their bounding volumes intersect those of other simulated entities and/or terrain features.

In their normal state, dynamic features on top of the terrain have the same effect as single state (non–destructible) features. All that is needed is the location and state of the dynamic object (to indicate the object's collision volume). Thus, a log crib or abatis will block a simulated entity's path.

The interesting situations occur when a dynamic feature has been destroyed. Destroyed features have different bounding volumes than their undamaged counterparts. Thus, a low flying aircraft which would collide with an undamaged building, may not collide with its destroyed version, given identical flight paths.

Collision detection must also be aware of defilade and ditch positions and orientations. For these DPFs, collisions occur with the ground. Collisions are reported when an entity enters a defilade from the wrong direction or hits the steep slope of a tank ditch.

4.4 Munition impact detection

Munition impact detection entails determining the intersections of a munition with objects along its flight path. For each intersection, the kind of object intersected as well as the location of the intersection are determined.

Munition impact detection differs from collision detection in that munition impact detection uses a line segment instead of a bounding volume to detect intersections with feature and simulated entity bounding volumes. The line segment represents the flight path of the munition during a small slice of time. Also, munition impact detection can prune out many features from consideration, since much of a munition's flight path may be above the highest feature in the area.

Munition impact detection has concerns with dynamic features similar to those of line of sight, since both trace a line segment to a specified target area. All DPFs must be considered to determine if they lie in the munition's path, with excavation berms presenting the most difficult challenge (see line of sight). Also, the state of a dynamic feature may determine whether or not it blocks the path of a munition.

4.5 Route planning

Route planning is the generation of a cross country route or a road route. Route planning does not consider low level obstacles such as individual buildings and trees, but rather high level obstacles, such as cities, forests, rivers, and mountains.

A *waypoint* is an x,y location that defines a desired destination along a route. A set of connected waypoints define a path. When used in route generation, waypoints define a general "area" as a destination, rather than a specific location, thus passing "near" a waypoint is adequate if obstacles prohibit access to the exact location.

Given a supplied list of waypoints, route planning modifies the waypoints to avoid obstacles, when possible; otherwise route planning indicates the route is not trafficable for the given waypoints. Route planning can also be used to verify that the supplied waypoints describe a trafficable route without being modified.

The only dynamic features which have an effect on route planning are bridges, tank ditches, concertina wire, and known minefields. Bridges must be considered because they can be destroyed (thus breaking an otherwise connected road network), and because AVLBs and ribbon bridges can be placed during the simulation.

Minefields can be large enough to reach beyond the scope of obstacle avoidance (see next section), and thus must be considered by the route planner. In addition, a minefield may contain a breached lane which could be used to obtain a shorter route distance compared to going around the minefield.

It is important to note that SAF terrain reasoning will only take into consideration those minefields which the routing force is aware of. In other words, an OPFOR unit will not be aware of a BLUFOR minefield unless the minefield has already been discovered by the unit or by another friendly unit. Thus, route planning may plan paths through undiscovered minefields.

4.6 Obstacle avoidance

Obstacle avoidance is used to determine an unobstructed path from one location (waypoint) to another. Low–level obstacles, such as other simulated entities and individual buildings and trees, are considered.

In their normal state, dynamic features on top of the terrain (e.g. log cribs) are handled in the same manner as single state (non–destructible) features; they are obstacles to be avoided. In the breached state, obstacles can be negotiated. Bridges, of course, can not be traversed when destroyed.

Obstacle avoidance must recognize hull defilade positions, covered machine gun bunkers, and infantry fighting positions in order to determine a path which enters them from the correct direction and to resist the usual predictive avoidance of the impending walls when seeking cover. However, obstacle avoidance should consider these positions as obstacles when not seeking cover. As with route planning, obstacle avoidance must take advantage of dynamically placed bridges in order to determine the most direct path to a target location.

4.7 Cover and concealment

Covered and concealed locations are positions which make an entity harder to see from a specified "enemy" eyepoint. An entity is "harder to see" when a greater percentage of its surface area is blocked as viewed from the enemy eyepoint. Covered locations are those where the blockage is provided by a solid material (such as the ground or a building) which also makes the entity "harder to damage" with direct fire from the enemy location. Concealed locations are those where the blockage is provided by a non–solid material (such as tree foliage), which does not protect the entity from direct fire. Covered and concealed locations which allow the entity to engage the enemy are generally preferable over those which merely hide the entity.

Cover and Concealment must be aware of all dynamic objects in order to intelligently determine advantageous positions. Destroyed buildings no longer provide complete cover, while excavated fighting positions are desirable when seeking cover. Additionally, tactical smoke can be used to provide concealment for entities when appropriate.

5. Dynamic environment issues

The need for correlation between the CCTT SAF and the visual display imposes a number of restrictions on both systems, due to the different nature of the databases used by each. While the visuals are concerned with polygons, colors, and textures to present the most realistic looking scene possible, SAF is concerned more with three dimensional surfaces and bounding volumes defining individual objects throughout the environment. The concept is analogous to "seeing" versus "feeling", causing the types of information stored in the two databases to be vastly different.

The sections which follow describe some additional challenges for CCTT with respect to the dynamic environment.

5.1 Destructible static features

The implementation of DSFs in CCTT requires the image generator to store each DSF in memory at all times. The amount of memory which can be spared in the image generator restricts the number of DSFs in the CCTT databases to 10,000. Since there are many more features in the database than can be represented as DSFs, decisions must be made as to which features will be destructible.

The first suggestion was to limit the DSFs to buildings, bridges, and possibly dams. Still, the number of possible DSFs exceeds the limit. One suggested solution is to allow only certain types of features to be destructible. A more likely solution is to specify certain areas of the database in which all buildings can be destroyed.

Another challenge to be dealt with is what to do when a vehicle is on a bridge and the bridge is destroyed. Should the vehicle fall, or be teleported instantly to the ground below? If the vehicle falls, the simulation of the fall may or may not involve orientation changes to the vehicle, but the "out the window" display must match the dynamics of the fall.

5.2 Dynamically placed features

Correlation between simulation databases and their corresponding visual databases has traditionally been extremely difficult to measure [8]. The correlation problem only gets worse with the addition of features which can be oriented and placed at run–time.

One of the hardest correlation challenges between SAF and the visual display is placement of tank ditches. A tank ditch can be composed of up to four segments. Each segment can be 30, 60, 90, or 120 meters long, and each has its own three dimensional orientation. The smallest terrain facet (excluding microterrain) is 30 meters, so it is obvious that situations can arise where a segment of a tank ditch will cross multiple terrain facets, and thus will not conform exactly to the terrain. The greater the difference in slope between polygon facets, the less the tank ditch will conform to the terrain.

The visuals handle this problem by modelling the tank ditch segments with "skirts" along either side. When placed, the areas of the ditch which do not conform to the terrain will show these skirts, giving the visual effect of dirt piled along the side of the ditch. SAF has the challenge of simulating the tank ditches in such a way that terrain reasoning functions like height of terrain and line of sight produce realistic results.

The placement of all DPFs, not just tank ditches, presents a challenge to SAF. No excavation should be placed across a river, or under a tree or building. An abatis or log crib should only be placed where it is realistic to do so (i.e. there should be enough trees in the immediate vicinity to support the feature). Also, because of their fixed sizes, log cribs and abatis must be carefully placed to sufficiently block the path if they are to serve their countermobility goal. Finally, relocatable bridges must be accurately placed such that they will adequately span the obstacle.

Breaches across DPFs are represented as independent models. Therefore, they must be individually placed, and, in the case of tank ditches and concertina wire fences, "snapped" to the proper location. The breach must then be recognized as taking precedence over the underlying DPF for mobility purposes.

Another topic being discussed is how to show incremental completion of DPFs to provide incremental results to combat engineering activities. To accurately show incremental completion of a task, many more models of DPFs in various stages of construction would have to be created and managed. A simpler solution has been proposed which uses existing models. For tank ditches, one segment of the ditch would appear at a time, after an appropriate duration, giving the effect of the ditch being built a section at a time. Log cribs could at first be placed below the terrain surface, and then raised over time. The log crib would appear to become larger in stages. Alternatively, a single model of "wood clutter" could be used to show incremental completion for a log crib or abatis; and a single "shallow defilade" model for all defilades.

5.3 Environmental effects

Environmental effects also present correlation challenges for CCTT. It is extremely difficult, if not impossible, to simulate what a person would see on the visuals based upon time of day, or when looking through rain, fog, haze, or smoke. The SAF database may contain an approximation of the formula used by the visuals to compute the density of these effects. This formula could then be used to degrade visibility in an attempt to emulate for a SAF entity's sensors what a trainee can detect.

6. Conclusion

The addition of dynamic attributes to a terrain database presents many challenges. Terrain reasoning in a dynamic environment becomes significantly more difficult and expensive. A tradeoff must be negotiated to allow run–time efficiency while maintaining the desired level of fidelity. The ideas presented here show CCTT's efforts toward this goal.

The terrain databases for CCTT are still being developed at this time. The information presented in this paper is based upon current understanding of the CCTT dynamic environment and how SAF plans to interact with it. Everything discussed herein is subject to change. However, a paper describing the final implementation, compared and contrasted with the issues presented here, is forthcoming.

7. References

[1] Lisle, C., Altman, M., Kilby, M., and Sartor, M., "Architectures for Dynamic Terrain and Dynamic Environments in Distributed Interactive Simulation", Proceedings of the Tenth Workshop on the Standards for the Interoperability of Defense Simulations, March, 1994.

[2] Li, Xin and Moshell, J. Michael, "Modeling Soil: Realtime Dynamic Models for Soil Slippage and Manipulation", 1993 SIGGRAPH Computer Graphics Proceedings, pp. 361–368, 1993.

[3] Cianciolo, Moureen E., "A Cloud Scene Simulation Model for Distributed Interactive Simulation", Proceedings of the Tenth Workshop on the Standards for the Interoperability of Defense Simulations, March, 1994.

[4] Campbell, Charles E., "Fluid Dynamics Methodologies for Computer Graphics", Master's Research Project Report, Computer Science Department, University of Central Florida, Orlando, FL, Fall 1991.

[5] Watkins, Jon E. and Provost, Micheline H., "Design of Terrain Reasoning Database for CCTT", these proceedings.

[6] "Close–Combat Tactical Trainer: Central United States", Database Design Document, Evans and Sutherland, Simulation Division, Salt Lake City, UT, February 28, 1994.

[7] Crowley, Jean M. and Moore, Ronald G., "Synthetic Environment Protocol", Proceedings of the Eleventh Workshop on the Standards for the Interoperability of Defense Simulations, September, 1994.

[8] Zvolanek, Budimir and Dillard, Douglas E., "Database Correlation Testing for Simulation Environments", 14th Proceedings of the Interservice/Industry Training Systems and Education Conference, November, 1992.

8. Biographies

Chuck Campbell is a software engineer with SAIC in Orlando, Florida. He has earned a M.S. in Computer Science from the University of Central Florida and a B.S. in Computer Science from Indiana University. He has over four years' experience developing Computer Generated Forces software. His interests include simulation, graphics, and computational geometry.

Gene McCulley is a software engineer with SAIC in Orlando, Florida. He is an undergraduate student at the University of Central Florida. He has two years' experience developing Computer Generated Forces software.

Design of Terrain Reasoning Database for CCTT

Jon Watkins and Micheline Provost
Science Applications International Corporation

Abstract

Terrain reasoning in CGF systems has traditionally been very expensive both in terms of time and space. Discussion of terrain reasoning concepts often focuses on algorithms, but the storage scheme for terrain data can also have a significant impact on performance. A number of challenges face CGF systems in this area including trading fidelity for performance, making efficient use of available memory, and data correlation with the visual display supplied to the man in the loop. The CCTT terrain reasoning database will build upon existing work and introduce new concepts to address these issues.

1.0 CCTT overview

The Close Combat Tactical Trainer (CCTT) is the first system in the Combined Arms Tactical Trainer (CATT) family of training systems. CCTT will utilize the Distributed Interactive Simulation (DIS) network protocol to provide a virtual environment for training of armor and mechanized infantry personnel. CCTT is composed of a variety of manned modules, an Operations Center (OC), Semi–Automated Forces (SAF), and several support workstations. The manned modules are cabin simulations with virtual out–the–window views for training on vehicles such as the M1A2, M2A2, and M113. SAF and OC provide emulated vehicles to populate the battlefield; they share a common architecture referred to as Computer Generated Forces (CGF). SAF provides a wide range of both BLUFOR (friendly) and OPFOR (enemy) entities. OC provides BLUFOR entities to support battalion staff training and to add depth to the battlefield with entities which provide resupply, maintenance, combat engineering, and fire support capabilities. Both SAF and OC are controlled via user interfaces provided on the SAF Workstations and OC Workstations, respectively. The actual simulation of the SAF and OC entities is provided by separate CGF processors dedicated to entity simulation.

There are three correlated databases used throughout the CCTT system: the visual database is used for all out–the–window visual displays; the PVD ("plan view") database provides a two dimensional plan view for display on user interfaces; and the "Model Reference" terrain database (or MRTDB) is used for all other terrain operations. MRTDB is designed first and foremost to support terrain reasoning operations on the CGF systems; however, the CCTT manned modules will also utilize this database for terrain functions such as collision detection, munition impact detection, and height of terrain. This represents a change from the original design which called for manned modules to receive terrain data directly from the image generator providing the out–the–window view.

2.0 Scope of paper

CCTT is utilizing spiral development to mitigate risk and provide operational incremental drops before contract completion. Simple terrain operations were provided in the initial drop of the CCTT SAF system ("Build 2"). This paper discusses the MRTDB format we propose to implement in the next system drop involving CGF components ("Build 4"). The MRTDB format implemented in Build 4 will be utilized by SAF Workstations, CGF, and manned modules. There will continue to be changes and improvements to the terrain storage mechanisms as system level issues are resolved and more CCTT components are implemented.

Build 4 functionality will include height of/above terrain, collision detection, munition impact detection, and line of sight. Additional functionality provided in Build 5 and beyond includes high level routing, obstacle avoidance, area intervisibility, cover and concealment, dynamic terrain effects, and weather. Database structures needed to support Build 5 operations are not discussed in this paper.

0-8186-6440-1/94 $04.00 © 1994 IEEE

3.0 Terrain storage issues

When considering possible storage mechanisms for CGF terrain databases, one must attempt to balance three driving requirements:

1. Data must be stored in a *compact* format to minimize caching operations at run–time and to minimize hardware requirements.
2. Ideally, the data is stored such that, once it is in memory, its run–time utilization is as *efficient* as possible.
3. Finally, the data must be stored with sufficient *fidelity* to meet system requirements such as fair–fight and correlation with visual displays.

In many ways, these requirements are mutually antagonistic. For example, storage of a point, radius, and height for tree geometries is compact, but may not provide the required fidelity. Storage of detailed filters which may reduce or eliminate operations supports run–time efficiency but increases storage requirements, and thus may increase cache misses.

4.0 Existing terrain representations

Three existing formats for CGF databases are referenced in this paper both as a source of ideas and to provide standards for comparison with the proposed CCTT MRTDB format: the Compact Terrain Database (CTDB) and Quadtree database formats presently used by ModSAF, and their predecessor, the original SIMNET SAF TDB. Throughout the remainder of this paper, these formats are referred to as MRTDB, CTDB, Quadtree, and SIMNET TDB.

SIMNET TDB: The SIMNET TDB uses a *patch* as the fundamental storage unit which represents all terrain data for a square area of terrain. Each patch is conceptually subdivided into 16 square *grids* and is composed of lists of vertices, edges, ground polygons, man–made structures, trees, treelines, and forest canopies. SIMNET TDB has a number of useful filters such as grid maps (which provide direct access to certain features by grid and type), grid masks (which provide rapid filtering of features based upon grid), minimum and maximum x,y,z values (which help eliminate features and/or patches from consideration), and patch guards (which can prevent caching of patches based upon a summary of patch data).

CTDB: CTDB [1][2][3] also uses the patch as the fundamental storage unit for terrain features, but it stores terrain skin information separately in *pages* of 4K chunks. The "compact" terrain database format lives up to its name by packing structures down to the bit level, storing the terrain skin as regularly spaced z–values referred to as elevation *posts,* and using a space saving fixed point representations that ModSAF refers to as a "fixed point basis". A given fixed point basis uses a fixed point number to represent some integral number of units, where the unit is referred to as the "basis". Some of the filters and organizational data present in the SIMNET TDB were eliminated in favor of space savings. CTDB is very successful at compressing data: the Fort Knox database is approximately 32M in SIMNET TDB format but only 4M in CTDB format. This means that there should be few if any cache misses even with limited memory.

CTDB represents a strong shift in favor of a compact representation while providing minimal loss to run–time efficiency. In some cases, compact representation was given priority over efficient "in–memory" operation as compared to the SIMNET TDB format. One example is the elimination of many filters; minimum and maximum elevation values are no longer stored by patch in CTDB. Another example is the reduction in organizational data; for instance, to process the linear features in a patch, it is necessary to iterate through all other patch features because linears are stored last and CTDB does not have indices into the feature lists by type. CTDB's use of elevation posts is both more compact and more efficient than the polygonal format stored in the original SIMNET TDB.

Quadtree: Quadtree [2][4] provides terrain feature information stored in subdivided areas of terrain (quads) and also stores road and river networks in a connected fashion well suited to routing. Quadtree stores features in a low–fidelity manner useful for "plan view" display and route planning. The CCTT PVD database format will be similar in many ways to Quadtree. MRTDB will utilize connected road networks similar to those stored in Quadtree for future design in support of routing.

5.0 CCTT challenges

The database formats discussed above provide a number of useful and innovative ideas for terrain representation. Because CCTT SAF is attempting to make maximum use of ModSAF in an effort to reduce development costs, it was initially believed that we would utilize a combination of ModSAF's CTDB and Quadtree representations with minimal changes. However, the sharp increase in visual system capabilities, as compared to SIMNET visuals, and the stringent system requirements for CCTT present a number of technical challenges for terrain database representation which required alterations and extensions to ModSAF's databases:

Terrain database density and size: The density of terrain features and resolution of terrain skin are being driven by the substantial capabilities of the ESIG–3000 which is the CCTT visual system [5]. The terrain skin will be represented at 30m spacing, increasing the storage space for gridded terrain posts by 17 times (as compared to 125 meters in most CTDB databases). Microterrain (also known as "cut and fill") will be used extensively. Detailed representations for ground mobility types and placement of point features (trees, buildings, bridges) will be much denser than in SIMNET. Indeed, feature densities will be sufficient to provide reasonable representations of urban areas. Storage is further impacted by the area covered by CCTT databases: 100 km x 150 km.

Visual database development is still underway and a number of fundamental design issues are still outstanding. However, initial estimates indicate that over 30,000 man–made structures (10,000 of which will be destructible) and over a ***million*** individual trees will be represented in the first full–size database. The SIMNET Fort Knox database has roughly 4,000 man–made objects and 25,000 individual trees in a database within one quarter of the CCTT terrain area.

Simple dynamic terrain: Some preplaced terrain features, referred to as destructible static features (DSFs) will change their geometry at run time, thus requiring a mechanism for uniquely identifying features and a means to efficiently alter their geometries. In addition, Combat Engineering features (referred to as dynamically placed features) may be placed before and during the exercise to provide survivability, counter–mobility, and mobility operations. Dynamically placed features may change the terrain skin, as represented in the static terrain database, as well as creating new obstacles where none existed before. Both destructible static features and dynamically placed features are discussed in [6] and [7].

Fidelity of representation: Because of the prohibitive computational load and storage requirements associated with detailed geometric representation of terrain features, it will be necessary to use approximations in some cases. With an eye toward the needs of future CATT systems, we have developed a database format that will permit variable fidelity representations with minimal storage requirement impacts.

Visual database storage techniques: The ESIG–3000 utilizes a number of techniques to reduce run–time database size, such as model libraries, basis sets and cluster features. We are investigating storage mechanisms which take advantage of these techniques without tying ourselves to the E&S visual database implementation.

New feature types: A variety of new feature representations found in CCTT required us to consider extensions to CTDB. For example, soil types in CCTT will be represented as areal features which need not conform to post boundaries, thus requiring us to store areals rather than storing soil types at elevation posts. This allows us to expand the number of soil types supported without increasing the number of bits required for posts. The first CCTT database will contain 31 soil types and these types may change at run–time due to rain. In addition, CCTT entities will detect collisions with tree trunks as opposed to foliage, while foliage effects line of sight, thus requiring us to store a radius for the trunk and a radius for the foliage; ModSAF required only one radius per tree instance. Hedgerows and walls are presently planned for the visual database; these are linears with both height and width. ModSAF required only linears with width (roads & rivers) and linears with height (treelines).

Many new types of terrain features are still under discussion at this time. These include penetrable forests, which are canopies with a high density of trees inside, and multi–level terrain such as bridges, overpasses, and tunnels.

Other Issues: In addition to the above issues which influenced our design decisions, other factors include Ada File I/O capabilities, improving caching performance, and support for different CCTT components such as SAF, OC, and Manned Modules.

6.0 MRTDB design and format

While considering various representation schemes to support the above requirements, we started with the excellent foundation provided by SIMNET TDB and ModSAF's CTDB. Future work will make use of QuadTree. We also drew ideas from the SIF standard, the implementation of the CCTT Visual Database, and Ada's data representation strengths and weaknesses.

6.1 Object–oriented design

We used an object–oriented approach in designing and implementing our terrain database [8][9]. We designed our database using Rumbaugh's object–oriented design methods and implemented it in the Ada language. Figure 1 is the terrain database Object Model using Rumbaugh's Object Model notation.

The following paragraphs give a brief description of the object classes shown in Figure 1, starting with the terrain database object class and followed by its component object classes. Object class names are capitalized within this section for clarity.

Terrain Database Class Description: This class represents the terrain database within a CCTT exercise, and is an aggregation of all of the objects which together form the Terrain Database.

Page Class Description: The Terrain Database is subdivided into square regions which are represented by this class. A given Page is further subdivided into a 4x4 array of Patches, and has header information as well as associated terrain skin (Elevation Posts) and terrain features (Terrain Features and Vertices). The Elevation Posts, Vertices, and Terrain Features for a given Page are stored and managed by the Terrain Cache. The Page header contains useful information about the Page including filters used to improve efficiency when performing terrain rea-

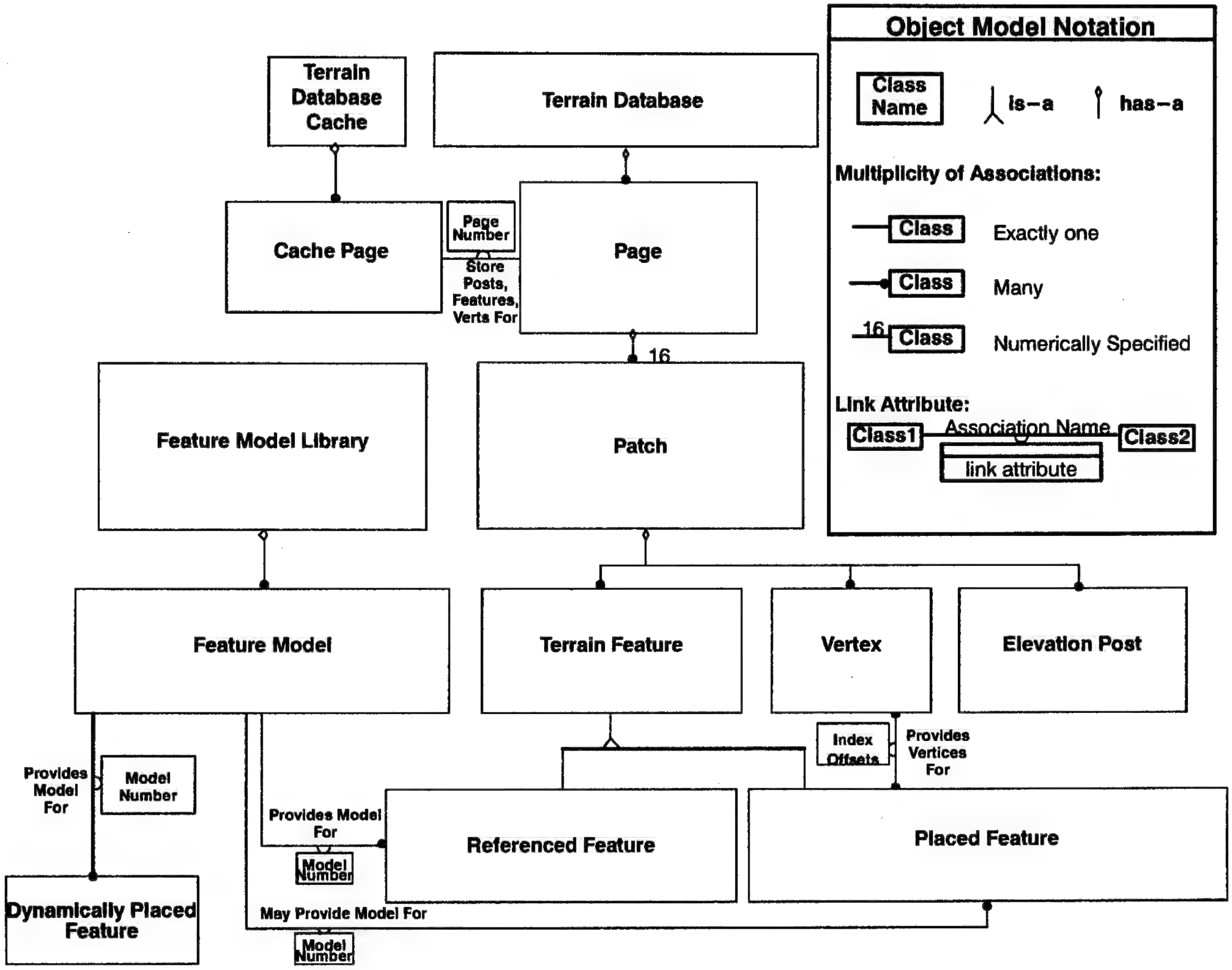


Figure 1. Terrain database object model.

soning algorithms, as well as information used when caching in from disk the Page's associated terrain skin and features.

Patch Class Description: This class represents a square region within a Page as described above. A given Patch is conceptually subdivided into a 4x4 array of grids. The Patch header maintains terrain reasoning filters as well as structures that allow the Patch to access its Terrain Features within its Page's list of Terrain Features. One set of filters maintained in the Patch header is the set of grid masks for certain types of features. For instance, a Patch's building grid mask indicates which of its grids are overlapped by one or more buildings. These filters vary from the CTDB format which stores this information at the post level.

Terrain Cache Class Description: This class represents the cache for the Terrain Database, and is responsible for making optimal use of memory when the memory available is smaller than the size of the Terrain Database. The Terrain Cache stores the Terrain Features, Vertices, and Elevation Posts for Pages, and additionally manages updates to destructible static features.

Feature Model Class Description: This class represents objects which maintain the information necessary to model trees, buildings, and linears, where a given feature instance references a feature model to complete its description. The Feature Models for a given Terrain Database comprise its Feature Model Library. Since use of models is the defining characteristic of MRTDB, the Feature Model Library is further discussed in Section 7.

Terrain Feature Class Description: This abstract class represents the terrain features and has two concrete subclasses, Placed Features and Referenced Features, described below. One attribute of a Terrain Feature is whether it is a destructible static feature. Each Page and Patch within the Terrain Database manages the Terrain Features which belong to/reside in their regions. Buildings and trees may cross Patch boundaries, all other Terrain Features may not. The "anchor patch" for a Terrain Feature which crosses Patch boundaries is the Patch which contains the Terrain Feature's "anchor" (a corner for a building, the tree center for a tree). Each Terrain Feature has a grid mask which indicates the grids it overlaps within its respective Patch.

Referenced Feature Class Description: This class represents those Terrain Features which reference a Feature Model to complete their definition. Unlike Placed Features, Referenced Features do not reference a Vertex List. An example of a Referenced Feature is a building which stores an anchor vertex, an orientation (stored using a fixed point basis), and an index to its building model in the Feature Model Library. The building model contains the remaining information necessary to compute the building's other vertices.

Placed Feature Class Description: This class represents those Terrain Features which reference a Vertex List. The term "Placed" comes from the fact that the Placed Features have each of their vertices explicitly defined (placed) in the Terrain Database when it is generated, as opposed to Referenced Features which have some of their vertices calculated at run–time. A Placed Feature may reference a Feature Model to complete its definition.

Dynamically Placed Feature Class Description: This class represents features which may be added to the Terrain Database at run–time. In addition, these features may be damaged, destroyed, or breached. Examples include tank ditches, tank defilades, log cribs, and abatis. These will be fully implemented in future Builds.

Elevation Post Class Description: This class represents terrain elevation posts, where the set of all Elevation Posts for a given Terrain Database defines the base terrain skin. The terrain skin may be altered by other objects such a microterrain and Dynamically Placed Features. A Post defines a square that sits parallel to the x and y axes, where the Post is at the lower left corner of the square. The other corners of the square are Posts that define their own squares. A line passes diagonally through a Post's square, either from Southwest to Northeast, or from Northwest to Southeast. Regardless of the direction of the diagonal, the Post is always at the Southwest corner of its square (allowing for a single post traversal algorithm instead of one for each diagonal direction). Each Post stores a diagonal indicator, a microterrain indicator, and its elevation using the fixed point basis described below for the Z value of a Vertex.

Vertex Class Description: This class represents (X, Y, Z) values within the Terrain Database. The (X, Y) values are in terms of a patch based fixed point basis. The Z values are in terms of a fixed point basis based on the range of possible elevations for the given Terrain Database.

6.2 MRTDB file format

The terrain database is stored in three separate files: a Headers file, a Feature Model Library file, and a Cache Page file:

Headers file format:
- Terrain Database header
- All Page headers
- All Patch headers
- DSFs by Page table
- DSF lookup table
- Wet–Dry Terrain Type Mapping

Feature Model Library file format:
- Number of Building Models
- Number of Tree Models
- Number of Linear Models
- Building Models
- Building Model Corners
- Tree Models
- Linear Models

Cache Page file format:
- Posts, Vertices, and Features for Page 1
- Posts, Vertices, and Features for Page 2

- Posts, Vertices, and Feature for Page N

7.0 Extensions and modifications to CTDB

Below we discuss some of the extensions and modifications to the CTDB format incorporated into the MRTDB format as required by the CCTT system. We begin with a discussion of the primary difference between CTDB and MRTDB: use of feature models in MRTDB.

7.1 Feature model library

The concept of the Feature Model Library was developed in an effort to meet one of the key challenges of CCTT databases: minimizing the overall storage requirements while maintaining the level of fidelity needed to facilitate fair–fight and to correlate with E&S's visual database. The Feature Model Library meets this challenge, as well as facilitating the implementation of destructible static features and providing MRTDB databases with additional flexibility and extendibility. The following discusses use of the Feature Model Library in Build 4; its functionality and usefulness will be expanded on in future Builds.

The Feature Model Library is a set of feature models, where each model has a unique model ID. Each model maintains information about a feature that is common across many features. For instance, a tree model maintains the foliage opacity, foliage height, foliage radius, and trunk radius for a particular kind of tree, for instance a Fir tree. Each Fir tree in the database can then reference the Fir tree model (via the model ID) to complete its definition, and since the model is stored only once, an enormous space savings can be realized. The table below provides some statistics.

Table 1. Comparison of MRTDB vs. CTDB: size of feature instances & total data stored.

Feature	MRTDB	CTDB
Tree	16 bytes, containing: Foliage radius Foliage height Trunk radius Foliage opacity	16 bytes, containing: Foliage radius Foliage height
Building (4 sided)	16 bytes, containing: Anchor Point Height 3 "placed" vertices State Unique ID	36 bytes, containing: 4 Vertices Height

The Feature Model Library facilitates the implementation of Destructible Static Feature since to "damage" a building we can simply modify the building model ID from the "normal model" ID to the "damaged model" ID. This is also much faster than performing computations to alter the geometry. Also, use of the library allows for a database modeler to supply the exact model of a damaged building, instead of describing the alterations that would need to be performed on a normal building to damage it. In addition, different buildings may be altered in different ways based on their damaged models, with no additional special case implementation.

Finally, since the Feature Model Library is stored in its own file, it may be swapped out for another library with the same feature models where some number of the feature models contain different information. For example, if we want all the trees to drop their leaves, we can read in a different Feature Model Library where all tree foliage opacities have been diminished. Furthermore, we can increase the fidelity of feature representations with little impact on storage requirements and with disregard for feature densities, since we only need increase the size of the models, not the size of each feature instance.

7.2 Header data

As stated in Section 4, CTDB strongly favors compact storage in contrast to storing structural or filtering data which is not strictly necessary. This design decision was invaluable for SIMNET databases which, in CTDB format, could be read entirely or largely into memory (unlike the larger SIMNET TDB format). However, CCTT's database size and density is such that storage of additional header data is inconsequential compared to overall database size. As a result, MRTDB page and patch headers contain a wide range of structural data. For example, both header types store the maximum Z value of any post or feature within their regions; this helps eliminate swathes of terrain from consideration, for instance when performing line of sight from a ground vehicle to a distant air vehicle.

MRTDB patch headers contain direct indices into their page's feature array. In order to determine if a given x,y point is on a road, one need only calculate which patch the point is in, then directly access the road linear features. CTDB requires the searcher to sweep through all other features in a patch, then process linears (both roads and rivers) in the search for a road.

Because of the large number of posts in CCTT databases, we were forced to compress the post size as much as possible; we consequently lost CTDB's "features present" bits which permitted rapid determination of which feature types could be found around a post. However, this data is retained at a grid level in our patch headers.

As CCTT implementation progresses, we may add abstract data to the headers, as needed. For example, we may need to classify patches and/or pages as "urban", "forested", or "rough". Also, the headers will be used to store information required to support simple dynamic terrain.

7.3 Independent subdivisions

Both CTDB and MRTDB grid masks store one bit of information for each of the grids in a patch. Because grid masks are stored in each feature instance to indicate which grids the feature overlaps, any change in grid mask size sharply increases database size and requires fundamental changes to all feature instances. Because SIMNET databases had 125 meter post spacing, CTDB was able to use a direct connection between grids and posts per patch (i.e. each post was also a grid). This provided a useful correlation between "features present" data stored for each post and grid mask values stored for each feature. CTDB grid masks contain sufficient bits to allow either 16 or 25 posts per patch (although 16 is declared as a constant in libCTDB). Because of the significantly higher density of posts in CCTT databases, we found it necessary to break the direct link between posts and grids which are the basic building block for the patch and page storage structures. All patches are composed of 16 grids regardless of the number of posts within the patch, thus the number of bits required for our grid mask need never change. Furthermore, the posts per grid side can be tailored for a given database providing additional flexibility when defining the page and patch storage structures.

Using CTDB's 4 posts per patch side in a CCTT database (30m posts, 100 km x 150 km) would require storage of over 1 million patches. Even with CTDB's minimal storage overhead (12 bytes per "patch group" of 4 patches and 4 bytes per patch), the page and patch headers would require 7M in CTDB format. Although MRTDB stores far more data in its headers, less than 3M would be consumed, given we define there to be 256 posts per patch (480m patch sides). When we create a correlated version of the SIMNET Grafenfels database, we will define there to be 16 posts per patch (500m patch sides). Thus, the patch sides can be tailored to match database–specific needs.

7.4 Other extensions

A number of smaller improvements deserve mention. The E&S Visual Database utilizes a mixed topology, which allows the hypotenuse of individual terrain facets to have either positive or negative slope. Mixed topology allows higher fidelity representation of terrain by allowing the terrain skin triangulation to match the direction of sharp terrain features. Both CTDB and MRTDB rely on triangulated post areas which must match the source data's hypotenuse (diagonal) for correlation. All posts in a CTDB database must have their diagonals going in the same direction; MRTDB maintains a bit for each post, thus allowing each post to have one of two triangulation schemes, thereby supporting E&S's mixed topology.

Both MRTDB and CTDB rely heavily on the use of fixed point bases, which require a tradeoff between range of values, accuracy, and number of bits available. CTDB utilizes a constant elevation range of +/–5000 meters for the elevation posts. MRTDB stores the actual range of values present in a given database and uses that for the fixed point basis range, thus insuring maximum accuracy for a given number of bits. Both CTDB and MRTDB use a 16 bit fixed point basis referred to as "patch units" for feature vertex x,y values. We have implemented "expanded" patch units which use 32 bit integers and store the same units as patch units, but allow for values outside of a given patch's boundaries. Thus we can store compact patch units in feature instances while storing expanded patch units in models, where use of additional space is not costly. This permits operations to exceed the normal range of patch units and permits use of model instances that cross patch boundaries. The latter is particularly important to us: CTDB splits patch–crossing features into multiple features, but we wished to avoid this due to our use of models and the desire to maintain the singularity of uniquely identified destructible static features which can be modified at run–time.

Ada file I/O limitations imposed a number of design constraints. For example, we cannot read in an entire page of posts, features, and vertices because these are different Ada types; we must either read each type separately or conduct expensive unchecked conversions when reading data from disk. However, Ada has provided some benefits besides facilitating encapsulation and maintainability; we have experimented with storing some data structures with sizes that do not align with 32 or 64 bit boundaries. Our preliminary investigations indicate that there is minimal performance impact when accessing such structures as long as they align to byte boundaries. Further timing tests will be conducted based on our current use of 24 bits per post (where CTDB uses 32 bits). A one byte savings initially sounds trivial, but CCTT databases will have well over 16 million posts. Thus, a byte savings translates to a roughly 16M reduction in data size.

8.0 Conclusions

The low–level representation of terrain data for use in terrain reasoning can play a major role in a number of factors vital to CGF systems: the run–time efficiency of terrain reasoning algorithms; the database's memory requirements and need for caching; the database's level of fidelity; and the database's correlation with other representations of the terrain, such as the visual database. We believe that the MRTDB format represents significant progress in each of these areas, while meeting CCTT's stringent requirements. We believe the strengths of this format (flexibility provided by models, object–oriented design, and compact representation) will not be fully realized until future CATT systems push requirements and needs still further.

9.0 References

[1] Smith, J. Compact Terrain Database Library User Manual and Report. *ODIN SAF Documentation*. March, 1992.

[2] Stanzione, T., Smith, J., Brock, D., Mar, J., Calder, R. Terrain Reasoning in the ODIN Semi–Automated Forces System. *Proc. of the Third Conference on Computer Generated Forces and Behavioral Representation*. March, 1993. pp. 317–326

[3] Texinfo documentation for libCTDB (libCTDB.info) contained in ModSAF 1.2 release.

[4] Stanzione, T. Terrain Reasoning in the SIMNET Semi–Automated Forces System, *Proc. of Symposium on Geographical Information Systems For Command and Control.* October, 1989.

[5] "Close–Combat Tactical Trainer: Central United States", Database Design Document, Evans and Sutherland, Simulation Division, February 28, 1994.

[6] Crowley, J., Moore, R. "Synthetic Environment Protocol", *Summary Report of the 11th Workshop on Statndards for the Interoperability of Defense Simulations*. September, 1994.

[7] Campbell, C., McCulley, G. "Terrain Reasoning Challenges in the CCTT Dynamic Environment", *Proc. of the Fifth Conference on AI, Simulation, and Planning in High–Autonomy Systems*. December, 1994.

[8] Meyer, B., *Object–oriented Software Construction*, Prentice Hall, Englewood Cliffs, New Jersey, 1988.

[9] Rumbaugh, J., Blaha, M., Premerlani, W., Eddy, F., and Lorenson, W., *Object–Oriented Modeling and Design*, Prentice Hall, Englewood Cliffs, New Jersey, 1991.

Session 1D:

Network Analysis

Traffic Characterization of Manned-Simulators and Computer Generated Forces in DIS Exercises©

Sandra E. Cheung and Margaret L. Loper
Institute for Simulation and Training
3280 Progress Drive
Orlando, Florida 32826

Abstract

Distributed Interactive Simulation (DIS) is an ambitious attempt to seamlessly integrate heterogeneous simulators of various fidelity levels via a communications network to allow them to interact in the same synthetic environment, by means of standardized messages, known as DIS Protocol Data Units (PDUs).

DIS traffic analysis has various purposes, one of which is capacity planning. This can be done effectively by understanding the traffic patterns of entities under specific maneuvers and interaction. In this paper, we characterize the traffic from DIS entities generated from computer generated forces (CGF) and manned simulators, in two cases. We will characterize this by the issue rate of Entity State PDUs, which comprises over 90% of DIS traffic. Our analysis will show that entities from CGF have a uniform traffic pattern and can therefore be used to populate a DIS environment effectively, while being able to plan the bandwidth required to sustain these entities.

1. Introduction

Distributed Interactive Simulation (DIS) is an infrastructure that enables heterogeneous simulators to interoperate in a time and space coherent environment. In DIS, the virtual world is modeled as a set of entities that interact with each other by means of events that they cause. Simulator nodes independently simulate the activities of one or more entities in the simulation and report their atttributes and actions of interest to other simulation nodes (as a point of reference, DIS is often used to simulate virtual battlefields, where the simulated entities are tanks, infantry fighting vehicles, combat aircraft, and infantry). Other entities in the virtual environment in turn are responsible for “listening” to the messages transmitted by other entities and determining which ones are of interest to them. These simulator nodes are linked by a communication network and communicate entity information using a set of common network protocols.

1.1 Simulation domains

DIS supports three simulation domains: virtual, live and constructive. The historical core of DIS has been continuous, real-time simulations, which have been designated as "virtual." These simulations include human-in-the-loop or crewed simulators and Computer Generated Forces (CGF). Virtual simulations are characterized by their requirement for real-time delivery, which is on the order of 100 milliseconds [3]. Because of the human in the loop, DIS assumes that exercise time corresponds with the actual progression of time.

DIS is also intended to interface with "live" simulations which include operational platforms and test & evaluation systems. Interactions between real weapon systems, sensors, and tactical communication links occur at much faster rates than virtual simulations, often less than one millisecond.

The last type of simulation is event driven wargames, called "constructive" simulations. Constructive simulations differ from the other two domains in that the simulation is at a higher level than that of a single entity. These simulations often move faster or slower than real-time. The intervals at

0-8186-6440-1/94 $04.00 © 1994 IEEE

which the states of all the participants are updated may be irregular and minutes may elapse between them.

1.2 Network traffic

As mentioned previously, DIS simulation nodes communicate using a set of communication protocols. The DIS protocol architecture, designed in the early 1980's under the SIMNET program, specifies a set of fixed data structures that are broadcast to every simulation node on the network. When high numbers of entities broadcast information, the DIS network begins to get congested. And as DIS expands beyond the bounds of SIMNET into virtual, live and constructive domains, and extends its capabilities to model things like human figures and dynamic environment, the network will become even more overwhelmed. Understanding the composition of this data and any patterns present in it become an important part of scaling to meet expanding DIS requirements.

This paper will focus on categorizing DIS network traffic created by virtual simulations, based on the type of simulator (manned or CGF). It will examine both manned and CGF entities to determine if they have regular traffic patterns. If so, this measure can be used to predict network requirements for exercises composed of large numbers of entities. Considering that future DIS exercises are expected to be comprised of up to 95% CGF [8], determining their traffic pattern could be an important part of scaling the network.

2. Virtual simulators

Virtual simulations fall into two categories: manned and CGF. Each has certain attributes which describe it and objectives that it brings to the virtual environment. The next sections provide a brief description of each.

2.1 Manned simulators

The primary object of manned simulators is that of training. The quality of training has been enhanced in recent years by technological advances. These simulators have the same look and feel of the actual device and/or environment being simulated. Another breakthrough in the training quality enhancement is the interconnection of traditional stand alone simulators via a DIS network. This allows training with other humans in the same virtual environment. For stand alone simulators (which pre-date the DIS era) to be able to take advantage of the benefits distributed simulator, an interface is often added as a front-end to make them DIS compliant. This interface unit then makes a non-DIS system capable of interoperating with other DIS simulators.

The majority of current DIS simulations are manned simulators. Manned simulators typically simulate a single type of entity (such as an M1 tank or an AH-64 Apache helicopter). These devices require a human-in-the-loop to operate the simulator, make decisions, respond to human commands, and interact with the environment (e.g., drive around trees).

2.2 Computer generated forces

Most DIS exercises have been and will be much larger than can be practically populated with human-in-the-loop simulators. Therefore, it is necessary to have many entities in the exercise that can operate under the loose supervisory control of human operators. Such Computer Generated Forces are capable of generating multiple entities (which may or may not be all of the same kind) and use vehicle behavior algorithms rather than humans to generate the actions of the simulated entities. Representation of platform level entities also includes the command and control heirarchy representing the missing human commanders.

CGF[1] have different requirements and capabilities from human-in-the-loop simulators, flowing largely from the differences in cognitive and perceptual abilities of CGF compared to human operators or commanders. For current or near-term systems, CGF entities are at the "platform" level with limited perceptual and cognitive abilities.

CGF form an integral part of team training, by providing effective opposing forces or by additional friendly forces in the virtual environment. In this way, the training effect is enhanced. The savings result primarily in the areas of personnel, hardware, facilities required to plan the exercises, setup, execution and analysis. In addition to training, CGF are also used for analytical purposes, and studies which involve large numbers of entities.

3. Network analysis

According to [2], future DIS exercises will scale to 100,000 entities[2]. Numerous sources have identified the network traffic to be the main bottleneck of future distributed simulation exercises, namely because the high orders of entities all broadcast their packets onto the network. In an analysis of the 1993 I/ITSEC demonstration, it was shown that over 90% of the DIS traffic was due to Entity State PDUs[1]. [However, a radio experiment conducted during the demonstration showed that 8 radio entities contributed to 16.5% of the total traffic! Similarly, an electronic warfare experiment showed that 42 emitting entities generated 15.3% of the

[1] In the DIS world, computer generated forces and semi-automated forces are often used synonymously when actually they refer to two separte functions. As defined in (Institute for Simulation and Training 1994b), Computer Generated Forces is the simulation of human entities, human controlled entities, and human command entities in the virtual battlespace by a computer. Semi Automtaed Forces are CGF controlled by an operator through commands and feedback not used in the real world.

[2] These entities will be a mix of live, virtual, and constructive.

total traffic. This confirms that as new capabilities are added to DIS, such as human figures, the associated traffic will grow exponentially]. Since over 90% of current DIS traffic is Entity State, this specific protocol data unit carries most of the burden for network congestion and therefore may provide insight into patterns associated with entity types.

There are ongoing efforts to reduce network traffic by filtering [5], compressing [6], multicasting [9], and using information handlers [7]. However, an additional means to scale networks is knowing a priori the traffic pattern of the exercise. This would allow the exercise manager to distribute resources as necessary, utilize systems more efficiently, and initialize appropriate groups. The following sections of this paper will address the issue of traffic characterization by analyzing data resulting from the 1993 Interservice/Industry Training Systems and Education Conference (I/ITSEC).

4. Traffic characterization

This section describes the data collection, trace contents, performance metrics, and analysis methodology by which the traffic of entities will be characterized.

4.1 Data collection

A total of 44 organizations gathered at the 1993 I/ITSEC for a period of 2 weeks in an effort to conduct DIS Interoperability Demonstrations. Each organization had its own individual network connection to a central hub (thus creating a star topology).

The data on the network was logged (the DIS data portion amounted to over 2.8 Gigabytes) and analyzed [1]. The data which was analyzed was collected by a single station on the LAN, and the assumption made is that the logger dropped none or minimal number of packets. Additionally, the logger restricted the logging to DIS PDUs only (logging the other network traffic would have been too overwhelming).

The purpose of the data logging was to provide overall I/ITSEC after action traffic analysis, and intended primarily to plan the capacity of future DIS exercises. For this purpose, the logger captured all the DIS traffic originating from the various DIS stations on the local area network.

4.2 Trace contents

The DIS traffic distribution, given in the average number of DIS PDUs per second, over the period of two weeks up and through the Interoperability Demonstrations is given in Figure 1. Note that the daily average for the first day (11/22/93) is quite misleading. The relatively high average obtained is due to the fact that the data logged spanned only a single hour, when all the stations were busy practicing PDU transmission.

The first of the two weeks spanning this effort was dedicated to debugging the network, and rehearsing the demonstrations. The traffic issued in this week was not very useful for further analysis.

The demonstrations took place on November 30, December 1, and 2. Each of them spanned no longer than a number of minutes. The rest of the time the network was dedicated to "free play" and for special interest experiments (such as electronic warfare, radio and network flooding). Of these special experiments, the network flooding test was particularly interesting and more on this analysis can be found in [1].

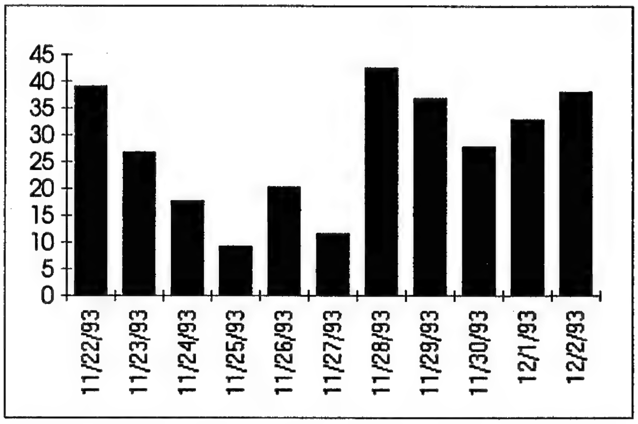


Figure 1. I/ITSEC Daily PDU Averages

4.3 Performance metrics

The performance metrics of interest to our analysis are:

1. Issue rate of Entity State PDUs per second for a CGF entity.
2. Issue rate of Entity State PDUs per second for a manned simulator entity.

The above four measures will be obtained for each specific maneuver defined below:

1. Stationary entity
2. Constant velocity (going in a straight line)

These measures will be obtained for an entity representative of each of the following venues:

1. Land (for example, an M1 tank)
2. Air (for example, an F-16)
3. Surface (for example, an FFG-7 frigate)

4.4 Analysis methodology

The data traces contained all the DIS traffic on the network, from all the entities which were active during the time the logging was performed. The primary tool used to obtain the results of our analysis was the traffic analysis tool written to obtain general DIS statistics of all the I/ITSEC data traces.

Determining which entities were generated from CGF and which were from manned simulators, was not possible only based on the PDUs. Additional information, such as the knowledge of pre-assigned IP-addresses and what each of these companies were participating with (type of simulator), was used in order to make the search for the proper data traces easier.

An additional difficulty in using the existing data traces was the applicability of the data to the type of analysis we want to perform. In the I/ITSEC data traces, entities were at best maneuvering in an adhoc fashion (or towards targets) and not necessarily in a very controlled manner. We were presented with the problem of wading through the data traces and find entities which were doing the types of maneuvers as defined in the performance metrics.

Not only was finding these entities and maneuvers difficult, it would be very unlikely that the comparison of the traffic issued by manned simulators and CGF would be fair. This is especially true for entities operating on or near the terrain database (such as land vehicles).

The data used for the analysis came from two sources: demonstrations data traces and compliance test data. It was not difficult to find stationary land and surface entities in the demonstration traces. The data collected as part of the DIS compliance testing, during the months preceding the demonstrations, however proved to be very useful. Not only was the finding of the entities much easier (most data traces contained only a single entity), but the particular maneuvers which we required were described in the test procedures.

5. Discussion of results

During the first (of a series of three) DIS demonstration (data logged between 10:00 -11:00 a.m. Tuesday, November 30, 1993) there were a total of 149 entities on the network. The distribution based on the type of entity, and whether they were CGF generated or from manned simulators is given in Table 1. As expected, most entities in the exercise were generated by CGF. The activity in the demonstration involved primarily ground battle (thus explaining the predominance in land entities), being fired upon by air entities. CGF entities are ideal targets, and an effective manner of populating the environment with a large number of unfriendly vehicles.

The third column in Table 1 gives the total number of Entity State PDUs (EsPDUs) issued by the entities of a particular category, in the entire hour which was logged. Solely based on this data, it is immediately apparent that most of the traffic came from the manned simulators, even though they do not outnumber the CGF entities. For each category of entity, the manned simulator entities issued a higher average of EsPDUs during that hour (with air entities having the largest average of 4744.3).

Type of Entity	Entities	EsPDUs
Land/CGF	95	53449
Air/CGF	10	5491
Surface/CGF	8	2824
Subsurface/CGF	1	2
Land/Manned	5	7894
Air/Manned	27	128096
Surface/Manned	2	1096
Subsur/Manned	1	2

Table 1. Entity Distribution

5.1 Entity selection

This section describes the manned simulator and CGF entities used to characterize the traffic of land, air and surface vehicles. Two different entity types per category (for example an M1 and a T72 for CGF/Land) will be selected and an attempt will be made to compare these with equivalent entities. In other words, ideally an M1 generated by a CGF will be compared to an M1 generated by a manned simulator. The selection of compatible entities also takes into consideration the type of dead-reckoning algorithm implemented (for our analysis all the entities had the same algorithm). Another condition placed on the analysis is that the underlying terrain database was the same for all entities, and comparable entities were placed at the same coordinates.

The choices of manned simulator entities (as could be derived from Table 1) did not vary in all categories. Land entities include launchers, which are typically not mobile and would not be interesting for traffic characterization. There was a better selection of air vehicles, ranging from fighter planes (FA/18s, F15s, F16s) to helicopters (UH-60s and AH-64s). One fighter plane and one helicopter was chosen for this analysis. On the other hand, there was not an abundance of crewed surface vehicles (in fact during this demonstration there were only two), and these happen to be the same ship (FFG-7).

The choices of CGF entities presented a similar problem. The problem with some CGF is that the entities may have a preprogrammed behavior, such as a built-in collision avoidance algorithm. In most cases, whenever a

CGF entity is called for, data traces from IST/CGF entities were obtained.

5.2 Stationary

This section describes the results obtained from the case in which the entities are stationary. The DIS standard specifies that when entities have not changed their state significantly (by changing locations or appearance) the issue rate of the EsPDUs should be set at 5.0 seconds (unless otherwise specified).

In Figure 2a the interarrival times of EsPDUs is given for several land entities generated by CGF, and by manned simulators. From this Figure it can be seen that the CGF entities adhere to the 5.0 seconds update rule, but that entities from manned simulators are issued at a higher frequency.

For air entities, it may sometimes not be within the capability of the simulator to remain stationary (unless parked on the terrain). Others have the capability of being frozen in mid-air (with a frozen non-zero velocity vector). Figure 2b shows the interarrival times of EsPDUs issued by 2 manned fighter jets, 1 manned helicopter, and 2 CGF helicopters. One of the manned jets had a peculiar behavior, and would produce two consecutive EsPDUs every 5 seconds (so in essence there would be periodic interarrival times of 5 seconds, 0 seconds, 5 seconds, 0 seconds, and so on). It is not known why this simulator behaved this way.

Surface vehicles generated by CGF have similar issue rates as those found for land entities. Their manned equivalent can sometimes be pretty accurate in their issue rate, though the sample size is smaller than that of land vehicles. The interarrival times for EsPDUs of 3 frigates are shown in Figure 2c. One of the manned FFG7 approaches the CGF created FFG7, whereas the other has a shorter interarrival time.

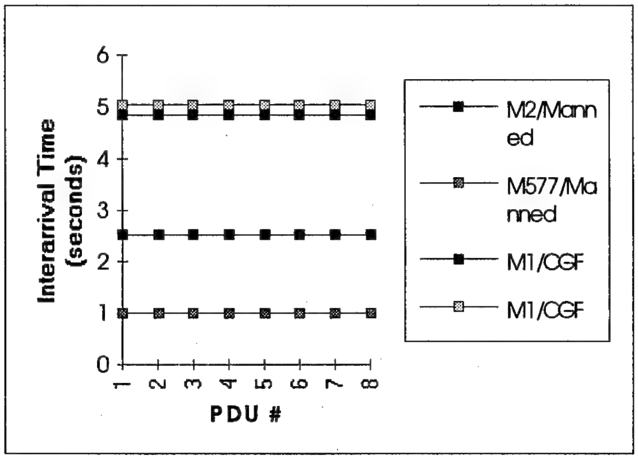


Figure 2a. Stationary Land Entities

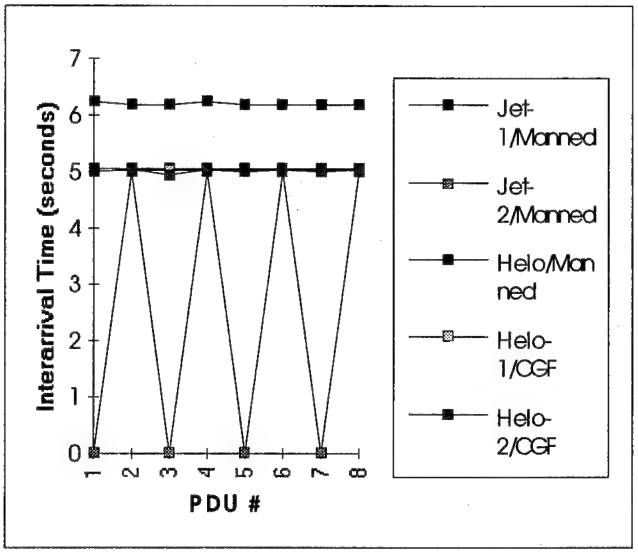


Figure 2b. Stationary Air Entities

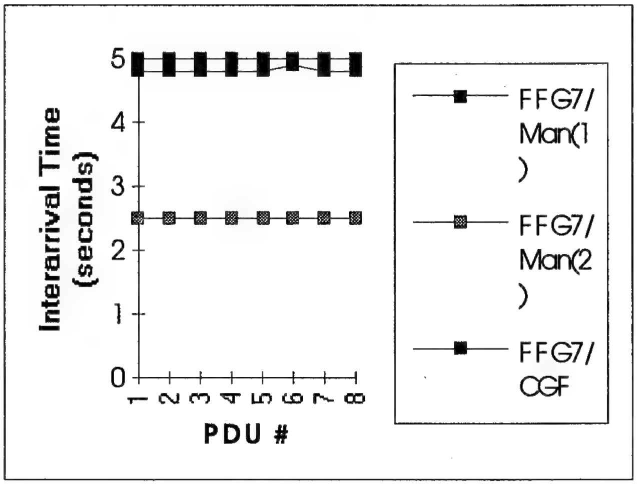


Figure 2c. Stationary Surface Entities

5.3 Constant velocity

This section describes the result obtained from the case in which the entities are proceeding at a constant speed. Again, the DIS standard specifies that in this case, due to one of the dead-reckoning algorithms, the EsPDU rate should be that of 1 EsPDU per 5 seconds (by using standard dead-reckoning algorithms, information that can be derived/extrapolated need not be transmitted, and thereby reduce the traffic considerably).

The same entities that were used above for the stationary case, are examined for their issue rates when going at a constant velocity. This may be impossible for some land entities which are permanently stationary (such as the M577 in Figure 2a). In particular, this proves to be a tremendous task for some manned simulators, because of the human-in-the-loop operating the vehicle. CGF entities, on the other hand, can be programmed to go at a given speed and to maintain this for a period of time.

The terrain becomes of importance in this test, since land vehicles operating on the terrain are subject to encountering hills and other obstacles, which in turn can cause the vehicles to accelerate (when going down hill) or decelerate (when going up hill or around an obstacle).

Figure 3a shows land entities generated by manned simulators and by CGF. Here, another manned simulator entity is used to replace the earlier immobile M577. For both the crewed simulator tanks, the speed did not remain constant. In one case, there was a terrain correlation problem and a flat piece of terrain was not found before logging the data. In the other case, the lack of constant velocity was primarily due to the human-in-the-loop factor. CGF land entities, however, showed a regular traffic pattern, albeit not always at the expected 5 second heartbeat rate. The velocity in these vehicles on the other hand, was maintained consistent throughout.

Figure 3b illustrates the case of air entities undergoing the same type of maneuver as described above. With manned simulators it was almost futile to present these figures, because it was difficult for the operator in the flight simulators to maintain a constant speed. Though terrain did not present a problem (they fly high above it) it was an added difficulty to keep the plane straight and level (no deviation in the orientation of the vehicle). The difficulty presented itself more with fixed wing vehicles (such as MIG-29s, FA/18s) than with rotary wing vehicles, which managed to maintain a constant speed and heading, though still issuing at a higher rate than desired (see Figure 3b). The same manned jet which in Figure 2b had a periodic issue rate, issued EsPDUs at intervals of 0.7 seconds and 0 seconds. This phenomenon is thus obviously not limited to being in a stationary or frozen state.

Finally, the surface vehicles which underwent the above mentioned maneuver are given in Figure 3c. The same manned FFG7 which had an EsPDU update rate of 2.5 seconds in the stationary case (see Figure 2c) managed to maintain a constant velocity but at an update rate of 0.5 seconds. The other manned FFG7 and the FFG7 produced by a CGF had equivalent update rates (5 seconds).

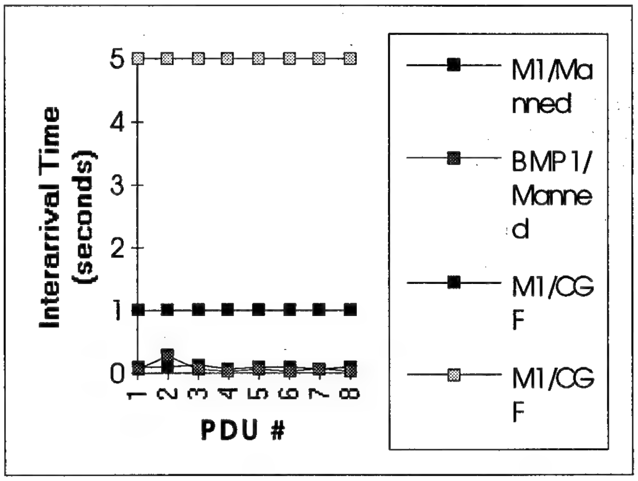


Figure 3a. Constant Speed Land Entities

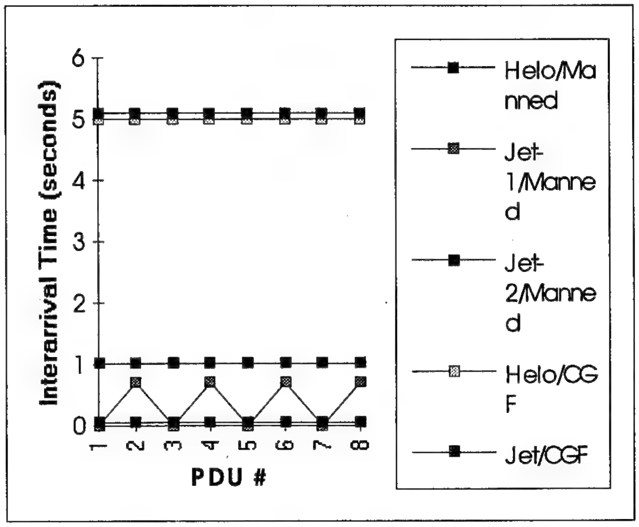


Figure 3b. Constant Speed Air Entities

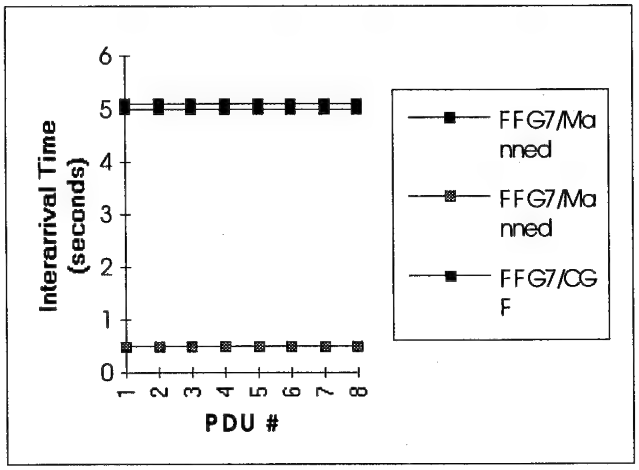


Figure 3c. Constant Speed Surface Entities

5.4 Discussion of results

The results presented in the previous two sections show the difficulty of obtaining traffic patterns for DIS entities. The characterization technique should be one which holds universally for all entities, which may be a problem (for example, not all entities can move).

The size of the sampled and analyzed population is far from ideal. One result prevailed throughout the analysis: traffic from CGF is regular and predictable (though not always consistently implemented - see Figure 3a, the two CGF M1s have different update rates). It is difficult to characterize manned simulators as an entire category for each implementation will vary (in the level of fidelity, for example). We were successful in obtaining traces for a minimal set of entities, operating in the same terrain, placed at the same coordinates, and modeling the same dead-reckoning algorithm.

6. Conclusions and future work

In this paper, the traffic from manned simulator entities and CGF entities are characterized, using data traces collected prior to and during the I/ITSEC 1993 DIS Interoperability Demonstration. From the data traces alone, one is, and should not able to tell CGF entities from manned simulator entities, but with prior knowledge of the types of systems brought to the demonstration and adherence to pre-assigned addresses, a traffic analysis tool is capable of filtering based on the identifiers of these entities and classifying them based on the type of simulator which issued them.

The results of the entity distribution (Figure 1) and stationary tests (Figures 2a-c) indicate that CGF entities broadcast fewer EsPDUs on the network than manned simulators. This is an important fact for large scale exercises which use CGF to populate the battlefield. Using CGF in the exercise can actually ease network congestion, compared to using manned simulators. Hence, a useful scaling tool for exercise developers.

There remains several more ways to further characterize the entities. The angular velocity can be maintained constant (yielding a turn at a constant rate). This paper was unable to show these results for the lack of adequate data. Manned simulators are hard to control when asked to perform specific maneuvers, keeping one factor constant without varying any others. CGF entities on the other hand, could be pre-programmed to have this kind of behavior and are not subject to direct human intervention. Another method of characterizing traffic from these two classes of virtual simulation entities is by a maneuver which involves a target, and weapons fire.

The analysis presented in this paper focused on the characterization of Entity State PDUs. At the present, over 90% of typical DIS exercises are comprised of EsPDUs, however future work should characterize emerging capabilities such as radio, emissions, dynamic environment and human figures, to see if similar patterns emerge.

Bibliography

[1] Cheung, S., “Analysis of I/ITSEC 1993 DIS Demonstration Data,” Technical Report, IST-TR-94-14, March 31, 1994.

[2] DIS Steering Committee, “The DIS Vision, A Map to the Future of Distributed Simulation,” Version 1, IST-SP-94-01.

[3] Institute for Simulation and Training. “Standard for Distributed Interactive Simulation - Communication Architecture Requirements”, IST-CR-94-15, May 1994.

[4] Institute for Simulation and Training. “DIS Lexicon,” September 1994.

[5] Vanhook, D. and Calvin, J., “Approaches to Relevance Filtering”, Proceedings of the 11th DIS Workshop, September 26-30, 1994.

[6] Vanhook, D. and Calvin, J., “PICA Performance in a Lossy Communications Environment”, Proceedings of the 11th DIS Workshop, September 26-30, 1994.

[7] Vanhook, D. and Calvin, J., “AGENTS: An Architectural Construct to Support Distributed Simulation”, Proceedings of the 11th DIS Workshop, September 26-30, 1994.

[8] Vrablik, R. and D. Wilbert, “The Use of Semi-Automated Forces to Simulate a 10,000 Entity Exercise,” Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, March 17-19 1993, pp. 169-178.

[9] Zyda, M., Macedonia, M., Pratt, D., and Barham P. 1994. Exploiting Reality with Multicast Groups: A Network Architecture for Large Scale Virtual Environments, Proceedings of the 11th DIS Workshop, Orlando, FL.

Acknowledgment

This research was sponsored by the U.S. Army Simulation, Training and Instrumentation Command under contract N61339-94-C-0024.

Realtime Data Analysis for the Joint Theater Missile Defense Simulation Network (JTMDSN)

Michael D. Gray, BDMESC
Craig K. Jones, BDMESC

Realtime data collection and analysis processes have been designed and an initial capability will be demonstrated for the July, 1994 JTMDSN test data. The JTMDSN is a one-year, Distributed Interactive Simulation (DIS) demonstration sponsored by the Defense Modeling and Simulation Office (DMSO). Detachment 4, 505 Command and Control Evaluation Group (CCEG), Air Warfare Center, is the project lead. Participants in this demonstration included the Theater Air Command and Control Simulation Facility (TACCSF) at Kirtland AFB, NM; the Navy Research, Evaluation, and Systems Analysis (RESA) facility at the Naval Command, Control, and Ocean Surveillance Center, San Diego, CA; the Space TACTICS Model (STM) and Proof-of-Concept Aerospace Defense Location (PADL) in Colorado Springs, CO; and the Theater Battle Arena (TBA) at the Pentagon.

The JTMDSN built on past TACCSF and RESA DIS accomplishments such as their War Breaker support, to make two more major contributions to the DIS community. First, Army, Navy, Air Force, and National systems interfaced via tactical data links using Signal Protocol Data Units (PDUs). Second, data was collected on-line from PDUs and reduced to records of key events which could be used to display mission performance and network performance measures in realtime. This paper focuses on this data collection and analysis capability. The JTMDSN DIS Gateway, including the data collection process, was designed and developed by Martin Marietta Corporation. Data analysis processes are being developed by BDM Engineering Services Company.

JTMDSN Operational Architecture

The Joint Force Air Component Commander (JFACC) at an Air Operations Center (AOC) was the highest command authority in this air defense exercise. The JFACC used inputs from tactical ballistic missile (TBM) sensors such as the Defense Support Program (DSP), the TPS-75 radar with missile tracker and correlator, an airborne Advanced Sensor, and AEGIS to prosecute both active defense of TBMs with Boost Phase Interceptors (BPI), PATRIOT, and AEGIS; and attack operations against transporter erector launchers (TELs) using Joint STARS, F-15E, and F/A-18 aircraft. The JFACC was assisted by a prototype fusion device for Launch Point Estimates (LPEs) and Impact Point Predictions (IPPs) developed by the PADL, called the Tracking Sensor Suite (TSS); by electronic emission exploitation inputs from PADL's NWARS model; and by JTIDS and a Constant Source terminal. A Control and Reporting Center (CRC), Army Air Defense Brigade, E-3 AWACS, Navy Battle Group, and E-2C provided lower echelon command and control, along with standard defensive counter-air (DCA) surveillance, identification and engagement capabilities.

DIS Architecture

DIS Gateways and other DIS capable models at each of the four sites communicated on a wide area network (WAN) using T-1 lines and DIS 2.04 protocol. The Gateways provided interfaces to models and environment generators at each site. Gateway processes included:

DIS Interface (DIS IF): Dead reckoning, Entity State PDU production, PDU receipt and routing, Event Report PDU production, PDU counting and reporting.

0-8186-6440-1/94 $04.00 © 1994 IEEE

Environment Generator (EG): Entity track maintenance and display, computer generated forces.

Data Link Interface (DL IF): Embedding tactical data link messages into Signal PDUs, extraction from Signal PDUs and routing of incoming data link messages.

Model and Simulation Interfaces (M&S IF): Provided environment to M&S; provided M&S status (Entity State), Fire, Detonate, Emissions, and Event Reports to DIS IF.

Trial Event Generator (TEG): Data extraction and recording.

Voice communications were also transmitted via Signal PDU. Entity State, Fire, Detonate, Electronic Emissions, Signal, Event Report, and Data PDUs were supported. RESA hosted all Navy assets. PADL/STM hosted DSP, NWARS, and the TSS. TACCSF hosted the AOC, CRC with missile tracker/correlator, AWACS, Army Brigade with PATRIOT and HAWK Battalions, F-15Cs, the Advanced Sensor, Joint STARS, and a copy of PADL's TSS fusion cell. TBA hosted the F-15E and the F-15C with BPI. Each site provided a portion of the computer generated threat forces and friendly background air traffic.

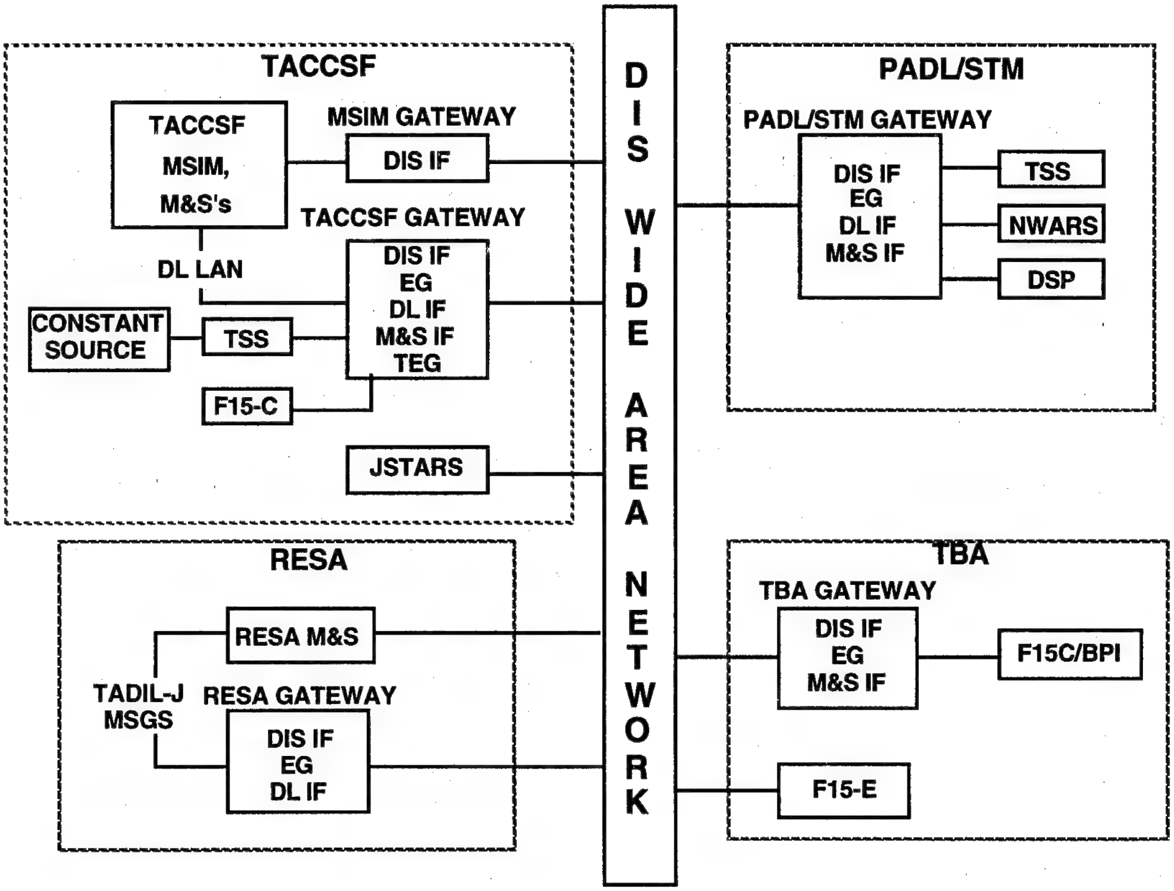


Figure 1. DIS Architecture

Tactical Data Links

Since TACCSF M&S design had already included the interchange of actual tactical data link messages, the DL IF was connected to the existing data link local area network (DL LAN) to exchange TADIL-J messages. At RESA, a TADIL-J message capability was developed for the fleet models and interfaced with the DL IF. At PADL/STM, the TSS development included the ability to read TADIL-J and TACELINT messages which were passed to it by the DL IF. NWARS and DSP sent TACELINT messages to the DL IF. The DL IF would embed up to 37 data link messages into a Signal PDU for DIS transmission and would extract and route data link messages from incoming Signal PDUs.

Event Report PDUs

The TACCSF MSIM GW picked existing CRC, PATRIOT, and HAWK target detection reports from the TACCSF track truth LAN and generated Event Report PDUs for DIS. Aegis and F/A-18 detections at RESA were supposed to be sent to the DIS GW for Event Report PDU production. Likewise, DSP detections and LPE/IPP reports, and TSS allocation orders were planned to be passed to the PADL/STM GW for Event Report production. (Thus far, no Event Reports from RESA or PADL/STM have been found in the data).

Operational Missions

Four missions were evaluated during the planned two-hour scenarios:

TBM active defense
TBM passive defense
Attack operations
Standard defensive counter-air

Measure of Performance

Measures of Performance (MOPs) are being computed for each of the four operational missions, such as:

Percent of TBMs killed
Timeliness and accuracy of TBM impact point predictions (IPPs)
Percent of TELs killed
Percent of penetrating enemy aircraft killed
Timeliness and completeness of sensor detections and reporting
Timeliness and completeness of resource allocations

In addition to these mission performance measures, MOPs are being accumulated on the DIS network performance, such as percent of PDUs received at each site, T-1 loading, and operator feedback on realism of the exercise.

Data Collection and Analysis

The PDU receiver and transmission processes of the TACCSF Gateway selected the following PDUs and forwarded them to the Trial Event Generator (TEG) process:

Entity State (create and remove entities, including kills)
Electronic Emissions (emitters on/off)
Fire (TBM launches, ordnance releases, engagement of targets)
Detonate (TBM impact, ordnance impact, missile outcomes)
Event Report (sensor detections)
Data (activity level, PDUs and PDU bytes sent/received each second)

The TEG used entity track files, which were maintained by the TACCSF environment generator, to obtain track truth information (location and entity type) on the source, target, and weapon associated with each event. A standard event record was then constructed and recorded by the TEG. Each event record contains the following information:

Time
Event type
Data source
Source identifier (site, application, entity)
Source true location (latitude, longitude, altitude)
Source DIS classification (kind, domain, country, category, subcat, specific, extra)
Source force ID
Target identifier
Target true location
Target DIS classification
Target force ID
Weapon identifier
Weapon true location
Weapon DIS classification
Weapon force ID
Target perceived position, heading, speed
Error ellipse description (for LPEs and IPPs)
Warhead/detonator type
Result/reason for action

This record provides an interface to the realtime data analysis processor. The analysis processor uses the operational performance event records to update summary tables which can be displayed to the analyst during or after a trial. A table is maintained for each of the four missions. Each table contains summary statistics which capture MOPs, and a time history record of performance against each target. The analyst can specify the entities of interest for each mission by selecting appropriate DIS classification values. The entities of interest for JTMDSN were as follows:

Threats: TELs, TBMs, enemy aircraft
Sensors: TBM detectors, LPE and IPP reporters, TEL trackers
BMC4I: TEL allocators, TBM impact warners
Weapons: TBM interceptors, TEL attackers, DCA aircraft and SAMs
Friendly aircraft (for fratricide measure)

Also, DIS network performance statistics were sent to the TEG each second via Data PDU. The TEG stored these PDU transmission and receipt counts, and scenario activity level counts in a file for posttest processing. A PV WAVE (Visual Numerics data analysis tool) program was developed by BDM to extract and display this PDU count data. The analyst can plot the PDU counts, T-1 volume counts, and number of active entities for a quick assessment of DIS network health over time. Future enhancements will enable plotting this DIS network health data in realtime.

Sample of Results

Table 1 shows the active TBM defense results from one 50 minute exercise. While this data was actually extracted after exercise completion, the realtime data analysis process has been designed to maintain a similar table while the exercise is running. These results are shown only to illustrate the output of the analysis process. They have no operational validity due to several factors, including: 1) model validation and accreditation was not part of this test; 2) most operators were not tactically qualified; 3) problems with immature software; and 4) the need to keep this report unclassified. This table identifies TBMs by their DIS Site.Application.Entity identifiers. Times are in seconds from the start of exercise. The need to automatically capture times of TBM entries into sensor and weapon system airspace was recognized, but not implemented for this exercise. Several TBMs were scripted to enter each of the PATRIOT and Aegis defended areas. The CRC's radar was in position to detect TBM trajectories toward the Aegis areas. TBM launches and engagements were collected from Fire PDUs. TBM impacts were collected from Detonate PDUs. TBM kills were evidenced by Entity State PDUs. PATRIOT and CRC detections were reported by Event Report PDUs.

ACTIVE TBM DEFENSE: TRIAL JBL08728											
	LAUNCH	AIRSP ENTRY		1ST DETECT		AIRSP ENTRY		1ST ENGAGE		KILL	TBM IMP
TBM	TIME	TIME	SENSOR	TIME	SENSOR	TIME	WEAPON	TIME	WEAPON	TIME	TIME
51.8.191	182	X	AEGIS			X	AEGIS				580.1
48.8.247	192	X	PAT	495.3	PATFU18	X	PAT	551	PATFU20	578.3	
48.8.248	202	X	PAT	510.4	PATFU18	X	PAT	544.9	PATFU20	584.5	
48.8.249	242	X	AEGIS	394.2	CRC	X	AEGIS	621.1	AEGIS		640.8
11.8.107	265.2	X	AEGIS	303.4	CRC	X	AEGIS				1101.4
51.8.192	362	X	PAT	682.3	PATFU18	X	PAT				756.9
48.8.250	902	X	PAT	1244	PATFU18	X	PAT				1304.6
52.1.41	913.6	X	AEGIS	1071.4	CRC	X	AEGIS				1315.8
52.1.42	963.6	X	AEGIS	1202.9	CRC	X	AEGIS				1362
52.1.43	1083.5	X	AEGIS	1108	CRC	X	BPI	X	BPI	1175.2	
11.8.108	1091.4	X	PAT	1616.7	PATFU21	X	PAT				1672.6
48.8.253	1202	X	AEGIS	1223.1	CRC	X	AEGIS				1601.1
48.8.271	1502	X	PAT	1828.9	PATFU18	X	PAT	1848	PATFU18	1880.5	
48.8.286	1922	X	AEGIS	1950	CRC						X
52.1.44	1983.5	X	PAT	2294.5	PATFU18	X	PAT				2382.2
52.1.45	2103.5	X	AEGIS	2263.2	CRC	X	AEGIS				2502.1
52.1.46	2703.5										
TOTALS	17	16		16		15		5		4	11

X: DATA MISSING

Table 1. Sample Actual TBM Defense Results

Table 2 shows the attack operations results for the same exercise. TEL launches and engagements were collected from Fire PDUs. TEL kills were collected from Entity State PDUs. TEL track updates, LPE reports, allocations of the TEL attack resources, and commitments of attack aircraft were collected manually. More automated data collection via Event Report PDU is being pursued for future DIS exercises.

ATTACK OPERATIONS: TRIAL JBL08728																					
	LNCH	MOVE	LAST TRK UPDATE			LPE REPORT			ALLOCATION				ATTACK AC COMMIT					AC UPDATE		ENG	KILL
TEL	TIME	TIME	TIME	ACCY	SOURCE	TIME	ACCY	SOURCE	TIME	ACCY	DELAY	RESOURCE	TIME	ACCY	DELAY	RANGE	ATTK AC	TIME	ACCY	TIME	TIME
51.8.150	182	482																			
48.8.240	192	492																			
48.2.241	202	502	780	1.7	JSTARS	236	0.6	ADV.SNSR	240	0.6	38	AWACS	685	1.4	483	66	F15E	860	1.8	1099.3	1108
48.2.242	242	542				249	14.1	ADV.SNSR													
11.8.62	265.2	565	840	0.7	JSTARS	270	1.5	ADV.SNSR	885	0.9	620	NAVY	X	X	X	X	X	X	X		
51.8.162	362	662				383	28.1	DSP													
52.1.29	913.6	1214																			
52.1.34	963.6	1264																			
52.1.37	1084	1384	1740	1.6	JSTARS	1290	0	DSP	1470	0.7	387	AWACS	1631	2.4	548	57	F15E	1998	1.1	2123.5	2131
11.8.66	1091	1391				1110	2.8	DSP													
48.8.246	1202	1502				1470	X	DSP													
48.8.224	1502	1802				1727	38.3	CORR													
48.8.225	1922	2222																			
52.1.38	1984	2284																			
52.1.39	2104	2404																			
52.1.40	2704																				
TOTALS	17	16	3	1.4		9	12.8		4	5.3	518		3	8.3	757	59		3	8	3	3

Table 2. Sample Attack Operations Results

Figure 2 illustrates one method for comparing PDUs sent with PDUs received, based on the PDU counts provided via Data PDUs. This data shows the RESA GW receiving all of the TACCSF GW Entity State PDUs except when the RESA GW was down. The MSIM GW received nearly all TACCSF GW Entity State PDUs until MSIM was terminated in the 45th minute of the exercise.

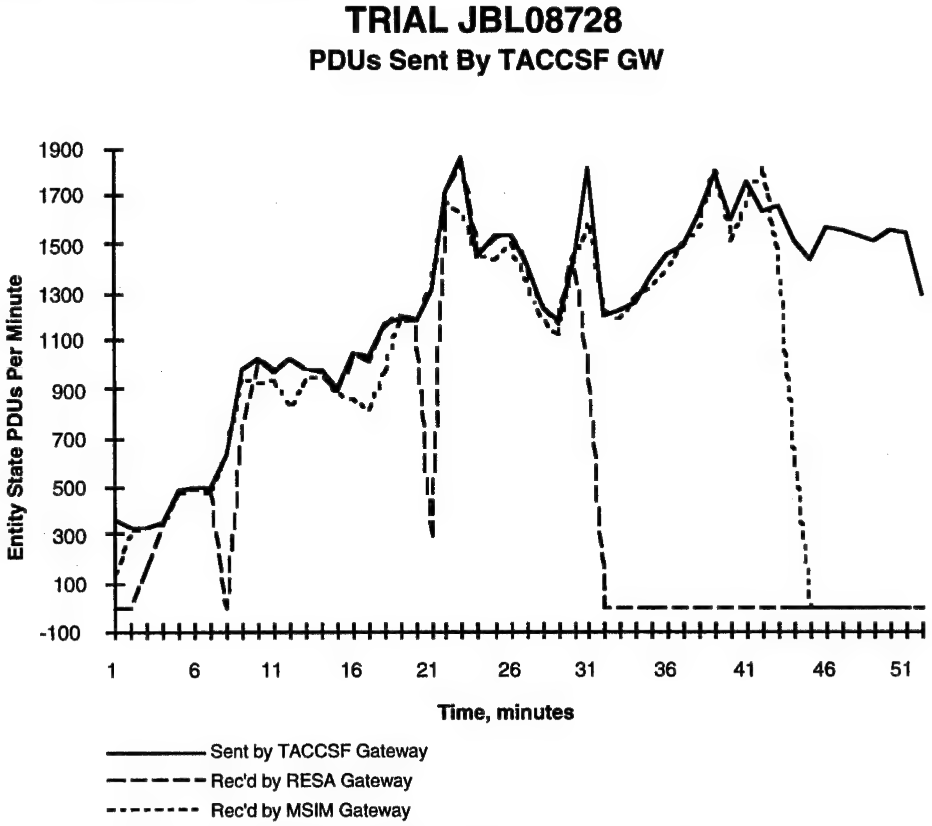


Figure 2. Entity State PDUs Sent and Received

Preliminary Results

Complete results of this experiment will be published in an October 1994 JTMDSN Test Report. Even though automated data collection from all models for all MOPs were not produced in the July 94 test, this realtime data collection and analysis structure provides the foundation for processing more detailed data from additional sources. In the future, much of the manually collected data will be replaced by automated collection methods or enhanced by providing on-line manual data entry methods. The JTMDSN Test Report will include details of the DIS architecture; and analysis of the tests' Critical Operational Issues based on data collected, operator feedback, and software engineer feedback. This Report will also include details of lessons learned and recommendations for future DIS projects, such as:

- Challenges in scheduling and testing in a large distributed environment. There is a need for better management of WAN hardware. System complexity causes delays in test initialization and frequent reliability problems.
- The need for early detailed specification of network design, including agreement on PDU implementation.
- The need for increased event reporting from network M&S's. Event Report PDUs should include entity true state information.
- The need for extended TEG functions, including event filtering, generation of periodic entity position reports, and generation of airspace volume entry/exit records.
- The need for better clock synchronization methods, both at the start and during the exercise.
- The utility of the PDU counting and reporting system.
- The need for DIS support of Identification Friend or Foe (IFF) models.

Session 1E:

Intelligent Agents

Insertion of an Articulated Human into a Networked Virtual Environment

David R. Pratt[1], Paul T. Barham, John Locke, Michael J. Zyda
Naval Postgraduate School

Bryant Eastman, Timothy Moore, Klaus Biggers, Robert Douglass, Stephen Jacobsen
SARCOS, Inc.

Michael Hollick, John Granieri, Hyeongseok Ko, Norman I. Badler
University Of Pennsylvania

Most Distributed Interactive Simulation (DIS) technology demonstrated in recent years has focused on vehicle interaction. The dismounted infantryman--the individual soldier--has been largely ignored or represented by static models. In six weeks of development, The Naval Postgraduate School, SARCOS, Inc., and University of Pennsylvania, under Army Research Laboratory sponsorship, demonstrated the insertion of a fully articulated human figure into a DIS environment. This paper describes the system architecture.

1.0 Introduction

The Simulation Networking (SIMNET) Project [4][3] connected low-cost, networked, man-in-the-loop simulations by a common protocol to simulate an armored battlefield. This emphasis was practical for several reasons, functional and technical.[2] The U.S. Army, the prime customer, was prepared for a Soviet land force in Germany. Experts agreed that the conflict, if it led to war, would have pitted armored units in a large-scale tank battle. In constructing SIMNET, this paradigm simplified decisions. For instance, a tank crew views the world though small windows. The largest of these has an 89-degree horizontal Field of View (FoV) and a much narrower vertical FoV [5]. Compared to the 180-degree-plus horizontal FoV of a human in the open, the attenuated view limits the computational load to process visual channels. Environment simulation was likewise restricted since crew members remained in the tank. Only the rough experience of being in a tank, and not the full detail of the surrounding environment, needed to be simulated for realism. Terrain databases could be constructed with only features influencing tank warfare.

Distributed Interactive Simulation (DIS) [2], the successor to SIMNET, likewise emphasizes vehicles.

As the New World Order evolved, emphasis shifted from large-scale tank battles to small regional conflicts which rely more on individual soldiers.[3] The Dismounted Infantryman (DI)

[1] Contact author at Department of Computer Science, Code CS/Pr, Naval Postgraduate School, Monterey, CA 93943; pratt@cs.nps.navy.mil; (408)656-2865; fax (408)656-2814

[2] BBN did a superb job with the existing technology. We in no way minimize their accomplishments.

0-8186-6440-1/94 $04.00 © 1994 IEEE

plays several roles in these conflicts, not all of which are currently feasible to simulate. However, some roles, like Special Operations Forces (SOF) or Military Operations in Urban Terrain (MOUT), lend themselves to simulation. These operations require small units of soldiers to act in close coordination. The team work used resembles the actions of civil police.

2.0 Protocol Representation of DI

SIMNET, as described in [1], was the first standard used in a distributed virtual battlefield. Since the modeled systems were primarily armored entities, the protocols and displays were optimized accordingly. The systems were limited to three basic types: Static (non-moving), Simple (no articulated parts), and Tank (two articulated parts, turret and gun). SIMNET humans were represented by two methods. In early systems, a texture map represented the soldier or fire team. Different postures (standing, prone, running, etc.) were represented by different textures. But when the figures moved they appeared to slide. In later systems, texture DIs were replaced by fixed models. Animations were created for running and crawling. Limited by fixed speed and stride, at different speeds they would "skate" on the terrain.

For DIS [4] articulations, each Degree of Freedom (DoF) has a 96-bit record, containing enumerations for articulation type, the changing parameter and value. While flexible for describing articulation, for entities with large number of DoFs, it is an expensive use of network bandwidth. Table 1 contains a length comparison between DIS and our optimized local method developed for the project.

3.0 Human Figure

For this project, we used the human figure model created by the University of Pennsylvania for their Jack Program. The model was converted to MultiGen Flight format to be compatible with the visual system, NPSNET-IV [6], allowing the model to be loaded by SGI's Performer API like other entity models.

The figure has 39 DoFs (Figure 1) in 17 joints. The torso has one joint at the waist. The neck has joints connected to the torso and head. Each arm has three joints: shoulder, elbow, and wrist. Each leg contains four joints: hip, knee, ankle, and toe.

4.0 System Architecture

The system architecture balances network loading, computational resources, and system requirements to optimize available equipment.

All network traffic used a single Ethernet segment, reducing the number of physical connections otherwise required by the number of point-to-point logical connections.

The design establishes two logical networks (Figure 2), one for point-to-point (TCP/IP) communications for optimized local formats, another for broadcasting (UDP/IP) DIS. The single physical network cut down the com-

Component	DI_guy	DIS 2.0.3	Difference
Header / Body	76	190	114
Articulations	156	624	468
Total	232	814	582

Table 1: Comparison of the PDU byte length to represent a 39-DoF Human Figure

[3] Of the three most recent major U.S. campaigns (Granada, Panama, and Persian Gulf), only one, the Persian Gulf, involved a large amount of armored vehicles.

putational resources, but it also limits future growth by maximizing potential system bottlenecks.

5.0 Component Functions

While each component is itself a complex system, our discussion considers the interface between systems, specifically network message formats. We will refer to the components of the DI_DISPLAY_DATA_MESS structure (Figure 3), the complete set of articulations and state data for the human icon.

5.1 ISMS VME Hardware Controller

The Individual Soldier Mobility System (ISMS) controller is a VME-based real-time computer whose primary functions are physical hardware control and monitoring of user input sensors. The user interface consists of three systems, the mobility platform, the sensor suit, and the head-mounted display (HMD).

The mobility platform resembles an exercise unicycle with a seat and pedals. The seated user controls the direction of the icon in the virtual world by swiveling the seat with his hips, and icon speed with pedal speed. The hardware applies pedal resistance based upon pedaling speed and terrain slope. The X-Y location of the soldier in virtual space is computed from pedal speed and seat torque.

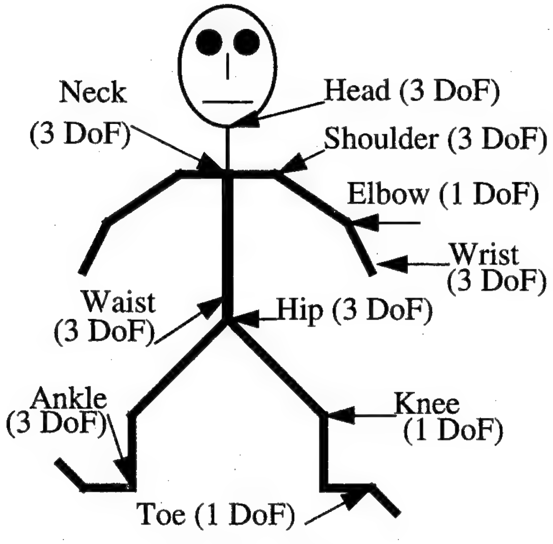


Figure 1. DoFs

The user wears a sensor suit, a number of limb attachments that output arm position and upper body orientation. The ISMS controller uses the raw input to fill in the ARM_ANGLES data for the arms and the waist position.

The HMD displays the virtual environment to the user and outputs head position. The HMD sensor data and the sensor suit input are used to compute neck and head position. Eye position, in world coordinates, is computed with an offset from the icon's world position and the computed offset to the head.

5.2 DI_guy

The ISMS/DIS interface (DI_guy) process is a communications server, elevation server, and a data display device. As a communications server, it dead reckons the human figure icons and formats DIS-compliant Protocol Data Units (PDUs). A copy of the terrain database provides the ISMS with elevation and slope for a given location. A primary use of DI_guy is to debug the system by showing current location, status, and parameter values.

The ISMS updates the DI_guy process at 60 Hz. DI_guy computes elevation from the X-Y and the terrain. The normal of the polygon the virtual soldier stands on is computed and given to the ISMS to compute resistive pedal loading. The data then passes to Jack to compute the remaining joint angles. Once Jack fills in the LEG_ANGLES for both legs, DI_guy forwards the data to the display devices.

5.3 Jack

Locomotion is computed from the global velocity vector and compass heading of the soldier. Current time is recorded at each footstep, and the time at each update determines the proper frame of the stride to display. A flag in the update packet indicates whether an entity is controlled by an ISMS operator, or some other source. If not ISMS-based, the figure's upper body is animated with a naturalistic arm swing.

Locomotion computations are only performed when the figure is in a standing posture. The posture can only change when the figure is not walking; thus a figure must stop to change posture, and stand up to walk. These restrictions avoid undesirable system behavior.

Additionally, a mechanism is provided for a forced stop. In normal conditions, the figure stops by slowing down and taking a final step when velocity drops to zero. Upon colliding with a fixed object, however, this behavior is unacceptable. A flag in the update packet indicates a sudden stop. When set, the figure returns to the default standing posture and the current step is canceled.

Upper body angles of the ISMS operator are measured by the body suit and sent to Jack, which performs simple validity checks. The angles are assigned to the corresponding joints.

A special case is the head/neck joint pair. These are not measured by the suit, but are derived from the viewpoint orientation (measured with a head-mounted sensor) and torso orientation. Since viewpoint orientation is in the global frame, the head/neck joints are adjusted so the simulated human's head orientation matches that of the viewpoint by subtracting the torso bend angles from the viewpoint orientation.

A correction is also done on the shoulder and head joints while the entity is prone, or undergoing a posture transition. Since the operator is always upright, not all measured joints correspond to the correct simulated posture. For example, if an operator firing a rifle goes prone (indicated by hitting a switch on the rifle), and raw joint angles are used, his arms will go *into* the ground since the simulated torso orientation is roughly parallel to the ground plane, and the simulated human looks into the ground. To correct, torso orientation is used as a correction factor for the shoulder and neck joints while prone (or in transition), thus the simulated soldier always has arms and head in the correct global orientation.

5.4 NPSNET-IV

The three display devices, the two HMDs and the Walk-In Synthetic Environment (WISE) (Figure 4) use NPSNET-IV [6], a 3D battlefield simulator, as the visual display tool. Since the soldiers can see the other non-ISMS entities in the simulation, they read the DIS network for the status of the other entities. The status of the ISMS humans goes over the point-to-point network.

For our demonstration (at Fort Benning), we used three variations of NPSNET. The first, the WISE, incorporated three large screen projection monitors (Figure 4), providing the user about 270-degrees FoV, each screen eight feet wide by six feet tall with 960x680 pixels. The low resolution produced minimal aliasing artifacts. Four speakers surrounded the ISMS, driven by a process that monitored the network and computed sound source location and strength. Together, the sound, force feedback from the ISMS, and the WISE display, produced a convincing environment for the user.

The WISE was only large enough for one solider at a time. The other two soldiers wore Kaiser Electrooptics HMDs with Polhemus head-tracking sensors. The two HMDs, one high and one medium resolution, provided immersive views of the environment.

The final display was the stealth platform that views the simulation without creating message traffic. The stealth sat in front of the soldier in the WISE and mirrored the network representation, allowing the soldier to verify upper body positions during hand and arm signals.

6.0 Network Implementation

With two logical networks, the visual systems have two paths to receive updates for each

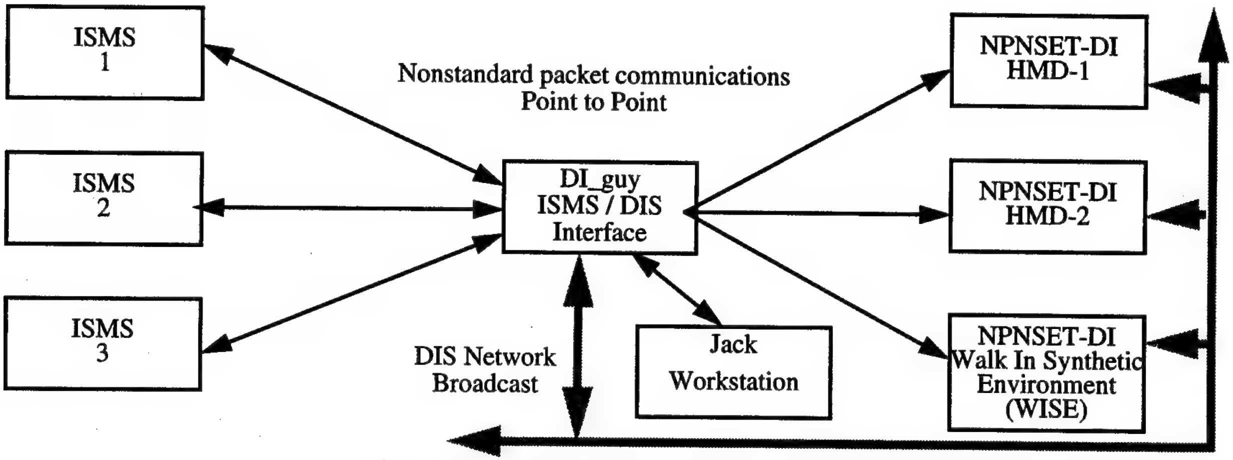

ISMS 1
ISMS 2
ISMS 3
Nonstandard packet communications Point to Point
DI_guy ISMS / DIS Interface
DIS Network Broadcast
Jack Workstation
NPNSET-DI HMD-1
NPNSET-DI HMD-2
NPNSET-DI Walk In Synthetic Environment (WISE)

Figure 2. Logical Network Connectivity

DI entity, one via the point-to-point connection, the fully articulated model, the second from the DIS PDUs put out by DI_guy. To avoid icon duplication, a filtering system was set up to discard DIS PDUs coming from DI_guy.

All ISMS subsystem internal connections are point-to-point with DIS communication broadcast. However, the drawback of point-to-point communications was that a message had to be sent to each visual system.

The basic message is sent from the DI_guy process to update the graphics process. As above, for ISMS systems, Jack fills in the lower body angles and checks upper body angle limits. The ISMS controller fills in the remaining information. On non-ISMS systems, Jack fills in all joint information and location is determined from algorithmic computations.

To avoid a needless conversion and "unconversion," we rejected the DIS round-world coordinate system for our 16X16 kilometer virtual area in favor of SIMNET's flat-world system.

7.0 Demonstrated Scenarios

At INCOMSS-94, we demonstrated a multi-soldier system using three scenarios. The first had two soldiers "dismount" from a ModSAF-controlled M-2 Bradley, run to a building and check for emptiness. The third ISMS represented an enemy soldier. Since weapons effects were not implemented, when the friendlies entered the building, the enemy ran out the back. The friendlies then returned to the M-2.

The second scenario was similar to the first except that the virtual building under assault corresponded to the actual building with our audience. At the end, one of the ISMS operators tossed a grenade though the door.

The final scenario had an ISMS operator give a series of arm and hand signals. These three scenarios mark the first time an articulated icon under human control has been shown in an DIS environment.

For the demonstration, the exercise network was divided into separate segments for the DIS and SIMNET protocols, connected by the LORAL PDU translator whose function was to convert DIS PDUs into the corresponding SIMNET PDUs and vice versa. On the DIS side, no noticeable delay between a moving ISMS soldier and the corresponding action on the DIS visual displays was observed. However, there was a consistent seven second delay for events on the SIMNET side.

To ensure consistent body orientation and posture, both ISMS and Jack updated the displays faster than the frame rate to account for the asynchronous nature of the SGI graphics pipeline and to achieve the least possible delay

```
TYPE ARM_ANGLES
   wrist[3]
   elbow[1]
   shoulder[3]

TYPE LEG_ANGLES
   toe[1]
   ankle[3]
   knee[1]
   hip[3]

TYPE BODY_ANGLES
   DOF_6_entity_origin
   DOF_6_view_point
   neck[3]
   head[3]
   waist[3]
   LEG_ANGLES left,right
   ARM_ANGLES left,right

DI_DISPLAY_DATA_MESS
   length
   type
   entity
   BODY_ANGLES body
   DOF_6_rifle_location
   velocity[3]
   status
```

Figure 3. DI_guy Message Format

between action and display. The ISMS sent out data as fast a possible--30-60Hz--to the DI_guy process, overwriting any pending messages. The same was done from DI_Guy to NPSNET. While this placed excess packets on the network, it did accommodate different process cycle times and reduced apparent latency.

8.0 Future Work

Efforts to insert an articulated human into the virtual world are just beginning. Following are some continuing and potential projects.

8.1 Ship Walkthrough

Ships represent one of the worst possible situations for a walkthrough. They possess the complexities of a building, and have to be smaller, self-contained, more intricate spaces. We envision two fundamental applications to ships. The first is human factors design. It is difficult to get a sense of reading instruments, say, aboard a swaying, heaving ship. The ISMS with a HMD can immerse the user in the environment with pedal force used to change ship motion. We could then determine the configuration of spaces and equipment. The second use would be familiarization. Ships have a large number of cables, compartments, piping, etc. A virtual ship model could be created. By turning systems off, such as the bulkheads (walls) and highlighting others, such as fire fighting systems, ship personnel could move through the environment to get a grasp of the layout.

8.2 Medical Corpsman

We have demonstrated the population of the world with icons moving under human control. A side effect of this capability in synthetic battlefields is that icons will be injured and require medical care. ARPA has started a program to train paramedics in the safety of the DIS environment. The basic capabilities of the medic are location of the wounded solider, wound identification and treatment, and triage.

8.3 Police Training

Many current day police officers require the same urban combat skills as the military. Increasingly, skills like hostage rescue, enemy identification, situation response, and team training are becoming a common part of police training. With the insertion of the human into the DIS environment, these skills can be practiced in simulation.

9.0 Conclusions

Fully articulated human figures can be incorporated in DIS, but work is required on articulation parameters. As shown in Table 1,

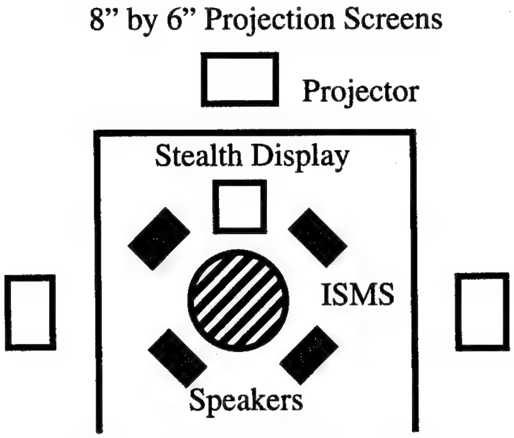


Figure 4. WISE Display System

582 bytes per message were saved by not using the DIS messages internally. By using a custom structure rather than the generic, the number of bytes needed to describe the articulations can be reduced by a factor of four. The differences in length can be attributed to a fundamental difference in purpose. For instance, we assume that soldiers will not change sides fifteen times a second, an eventuality that is fully accounted for by the DIS protocol.

Due to the number of articulations and human motion complexity, systems can be expected to send packets at the frame rate. Assuming 15-Hz, and considering only packet size, each soldier produces the network load of five to eight tanks or three high performance aircraft, potentially crippling a large scenario.

The computational load of Jack and the DIS conversation process did not prove excessive. Michael Hollick and John Granieri of UPENN have developed a table-driven Jack more suited for low resolution display of human figures. Bryant Eastman and Tim Moore of SARCOS have placed the DI_guy functionality into a heavily modified NPSNET and will be porting the table-driven Jack onto the VME realtime system in the ISMS. These enhancements represent a significant reduction of the number of machines and packets required for the articulated human.

10.0 Acknowledgments

This project was made possible by the Human Research and Engineering Directorate, Army Research Laboratory (ARL). Not only were they project managers, they originated the idea, provided technical guidance for the ISMS, and lent the insight to form the NPS/UPENN/SARCOS team. We could not have done it without them. We would like to thank Farid Mamagahni and Jim Madden for organizing the Fort Benning demonstration. The terrain database was developed at the Topographic Engineering Center (TEC), Fort Belvior under the guidance of George Lukes and Jay Banchero. The HMDs were provided by Kaiser Electrooptics with assistance from Frank Hepburn.

11.0 References

[1] Pope, Arthur, “SIMNET Network Protocols,” BBN Systems and Technologies, Report No. 7102,Cambridge, MA, July 1989.

[2] Institute for Simulation and Training, “Protocol Data Units for Entity Information and Entity Interaction in a Distributed Interactive Simulation (DRAFT),” Version 2.0.3, IST-PD-90-2,Orlando, FL, May 1993

[3] Thorpe, Jack A., “The New Technology of Large Scale Simulator Networking: Implications for Mastering the Art of Warfighting,” Proceedings of the 9th Interservice/Industry Training System Conference, Nov. - Dec 1987.

[4] Institute for Defense Analysis, “SIMNET,” Draft, Arlington, VA., May 1990.

[5] Perceptronics, “SIMNET M1 Crew Manual,” SIMNET Manual No. PTUM 001-1250-89-10 (rev-2), Woodland Hills, CA.

[6] Macedonia, Michael R. Zyda, Michael J., Pratt, David R., Barham, Paul T. and Zeswitz, Steven, “NPSNET: A Network Software Architecture for Large Scale Virtual Environments,” Presence, Vol. 3, No. 4. Fall 1994.

[7] Badler, Norman I., Phillips, Cary B., Webber, Bonnie Lynn, “Simulating Humans,” Oxford University Press, New York, 1993.

Extending DIS for Individual Combatants

Douglas A. Reece
Institute for Simulation and Training
University of Central Florida
3280 Progress Drive
Orlando, FL 32826
dreece@ist.ucf.edu

Abstract

The domain of DIS military training simulators has recently been expanding to the level of individual combatants. After gaining some experience with an individual-level simulator, we have identified several areas of the DIS protocol that need to be expanded or changed to accommodate individual humans. In the Entity State PDU, some information in the Entity Type and Appearance fields should be replaced with detailed upper body, limb and weapon position information. An intelligent human figure animation algorithm should be used to dead reckon the lower body. Weapons and other objects should gain status independent of the entity through the use of a modification of the Destructible Entity protocol. Finally, the Fire PDU should indicate the scatter pattern and direction vector of a burst.

1. Introduction

Various users of military training simulators are becoming interested in creating simulations for individual combatants. In order to take operate with other simulation platforms, these systems will be made compliant with standards for Distributed Interactive Simulation (DIS). While the existing DIS standard actually has a mechanism for representing a single person, this mechanism is crude and is really only an afterthought to the primary goal of representing tanks and other vehicles. The Institute for Simulation and Training (IST) is currently involved in a project to develop a training simulator for individual soldiers in an urban combat environment. As part of this project we have identified several areas of the DIS design that are inadequate for such a simulator. In this paper we discuss several problem areas and possible solutions. We first describe the training simulator that provides the motivation for this study of DIS standards. We next discuss some of the problems with the existing Entity State Protocol Data Unit (PDU). We have found that the biggest shortcomings are in the description of body position and the representation of objects; the next two sections thus discuss body position and object representation. The final section discusses DIS extensions required to represent small arms fire in individual combatant simulations.

2. A simulator for individual combatants

IST is currently participating in the Team Target Engagement Simulator (TTES) project under sponsorship of the Naval Air Warfare Center Training Systems Division. The TTES system will be a training simulator for small units. The application domain will initially be Marine Corps units in urban terrain, but could potentially be expanded to Special Forces operations, hostage rescue missions, etc. IST's role in the project is to develop computer controlled hostile and neutral entities.

TTES is intended to meet specific requirements for simulating combat on an individual level:

- The simulation platform must be reasonably compact; trainees will be confined to a limited area.

ISBN 0-8186-6440-1. Copyright (c) 1994 IEEE. All rights reserved.

0-8186-6440-1/94 $04.00 © 1994 IEEE

- Trainees should be able to stand, crouch, fall prone, and make other small movements to take advantage of small but tactically significant terrain features.
- Trainees should be able to use their weapons in as natural a way as possible.
- Body positions and movements should be represented and displayed in enough detail to allow trainees to recognize important states and actions. For example, whether another entity's weapon is deployed for use, where the entity is aiming the weapon, whether the entity is picking up or throwing an object, and where the entity is looking. Eventually, the system might allow trainees to communicate with arm signals.
- Friendly, hostile, and neutral entities should be represented.

In the TTES system, an individual soldier will stand in front of a large projection screen (or screens) on which is displayed his view of the virtual environment. Foot switches allow the soldier to move and rotate his virtual body in the simulation. Position sensors on the soldier's head allow him to move up and down, translate, and reorient his virtual body without using the foot switches. This configuration allows the trainee to make small translational movements without using the foot switches. Sensors on the soldier's weapon detect the aiming point and allow marksmanship training.

3. The Entity State PDU

The DIS protocol specification (version 2.0.4 [1], [2]) describes the Entity State PDU. As with many other aspects of DIS, this PDU was based on an original design that addressed tank warfare. Several fields of the Entity State PDU have shortcomings for individual combatant simulation.

3. 1. Force ID and Alternate Entity Type

These fields were originally included to allow entities to be displayed differently depending on their Force ID. For example, force 1 trainees would see other force 1 entities as US M1 tanks, but force 2 entities as T-72 tanks; meanwhile, force 2 trainees would see force 1 entities as T-72's and force 2 entities as M1's. Unfortunately, this arrangement is highly unrealistic for any case where the opposing entities do not have similar characteristics. For example, the soldiers on one force may have more automatic weapons than the soldiers on the other force. Some soldier will thus appear to one force to have an automatic weapon while to the other force he will not. Another problem with guises is that neutral entities may have to treat different forces differently; the symmetry of appearance is thereby destroyed. Since these fields of the PDU are not useful, they can be ignored or eliminated.

3. 2. Entity type

This field is intended to represent static type information rather than dynamic state information—for example, whether the entity is a life form or a vehicle. This information is used to generate an image of the entity. For individual soldiers, however, much of the information is dynamic. The *domain* subfield (land, air, water, etc.) depends on the current location of the soldier. It is not really meaningful for individuals and can be ignored. The *subcategory* identifies the [one] weapon carried and is inadequate for describing the soldier's load. As we describe more below, this subfield should also be ignored in favor of a more complete representation of simulation objects such as weapons. The *country* could change if the soldier changed uniforms; perhaps it should be replaced by a static ethnic type and a dynamic "clothing state." Other subfields of the entity type field may be left as is.

3. 3. Appearance

This field is intended to contain dynamic state information describing the observable appearance of the entity. Part of this field contains vehicle-specific information and can be ignored. The life form specific information includes posture state and primary and secondary weapon status. "Posture" state is sometimes actually posture (standing, kneeling, prone), and other times activity (swimming, parachuting) or movement (walking, running, crawling, jumping). This set of states is not sufficiently rich for a detailed simulation; it lacks important body positions (such as crouching below cover, or leaning to peek around a corner) and arm positions (to indicate weapon positions, give signals, allow throwing motions, etc.). Simulations for individual combatants will eventually replace postures with limb positions (or joint angles). These new human articulated parts will require new enumeration values in the DIS standard. Postures may be retained for lower fidelity simulations that do not need to generate detailed images of human figures.

The weapon status portion of the appearance field allows three states: stowed, deployed, and in firing position. As with posture, an individual combatant level simulation will require much more detail. For example, the immediate threat of a situation may depend on whether an enemy soldier is aiming his weapon at the trainee or elsewhere. While it may be possible to compute weapon position from detailed arm positions, we believe that a new record should be added to describe the position and orientation of the primary weapon.

The secondary weapon status part of the appearance field is not currently useful, because the DIS standard only allows only the primary weapon type to be specified in the entity type field. We discuss how to expand the treatment of secondary weapons as objects below.

3. 4. Location, velocity, and orientation

The DIS standard specifies that the reference point for an entity's location is the center of its bounding volume, excluding its articulated parts. This definition works neatly for vehicles, which are rigid, but creates complications for flexible human entities. We will assume that the upper torso is the core component of the human entity.

When a human changes postures, its bounding volume and the center point move. This correlation of posture and location makes it difficult to maintain an accurate display of a human's position. Consider the standing soldier falling prone in Figure 1. While his location—his upper torso—moves forward and down continuously, at some point his state changes instantaneously from Standing to Prone. While he is still Standing and his location is moving down, it would appear to remote entities that he is sinking into the ground. Dead reckoning algorithms that use velocity exacerbate this problem.

The discrepancy between torso position and posture can provides even more motivation for eliminating postures in favor of detailed limb positions in the Entity State PDU. For lower-fidelity simulations, however, it is still desirable to use postures. To avoid the possibly frequent location changes that cause erroneous displays on remote simulators, we propose to add new low-fidelity location, velocity and orientation fields to the PDU. These would indicate the status of a rigid human figure whose location, velocity and orientation did not depend on posture. The existing location, velocity and location fields would indicate the true status of the upper torso. In our prototype system we are using a human figure model that uses a foot as the origin; this is equivalent

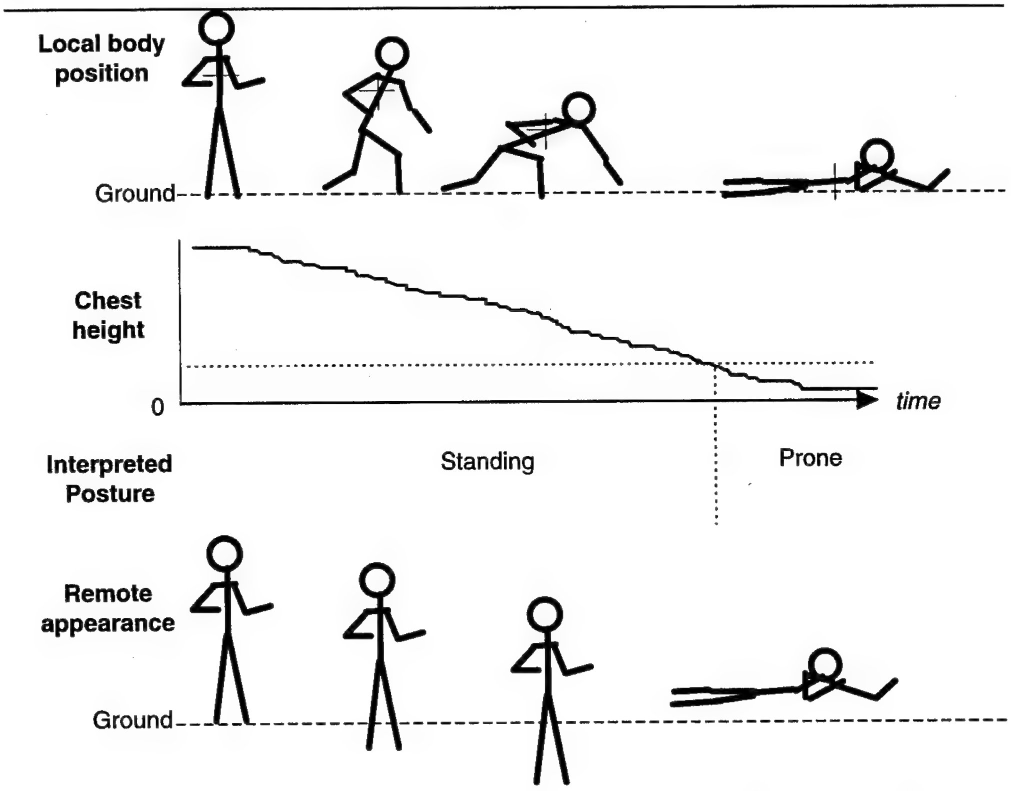


Figure 1. Errors in apparent position of entity due to correlation of location and posture.

to using the upper torso if all body-part transformations are known.

4. Position representation

The examination of the Entity State PDU above revealed the need for an improved representation of body position for human entities. There are several aspects to this problem arising from the fact that a DIS system has several different representations for a simulated entity. Figure 2 shows these different representations. The trainee himself is the true position; sensors provide a measured position for the simulator platform; the platform forms a DIS position, which is transmitted to other platforms; remote platforms used dead reckoning algorithms to interpolate the trainee's position; and the image generator produces an image in the virtual world for the remote entity. In each transformation between representations there is an opportunity to take measurements, make abstractions, encode or decode data, sample the data, interpolate between samples, add noise, filter noise, compress or expand data, introduce delays, decode etc.

4. 1. DIS representations

The first representation is the human trainee himself. His or her position must be measured by sensors to create representation 2. The completeness, accuracy, and frequency of position sensing determines what information the rest of the system has to work with. The simulator platform itself uses the measured position to recognize movement actions, determine the viewpoint for the image generator, to detect collisions, detect throwing actions, and perform any other modeling tasks necessary. The simulator may infer some information rather than measuring it directly; for example, from upper torso position and hand position, the arm position may be inferred.

The simulator platform must also create the DIS representation of the soldier's position. With the current standard, this means that measured body position must be abstracted to entity location, orientation, posture and other appearance characteristics. In the previous section we described some of the problems with this representation.

Remote simulators use a constant velocity or constant acceleration dead reckoning algorithm to compute position and velocity (representation 4) between Entity State PDU transmissions. The source vehicle simulator only sends out Entity State PDU's when the error between the dead reckoned position and the true position is greater than some threshold.

Finally, the encoded appearance information is decoded by the image generator to produce a detailed image of the original entity (representation 5)

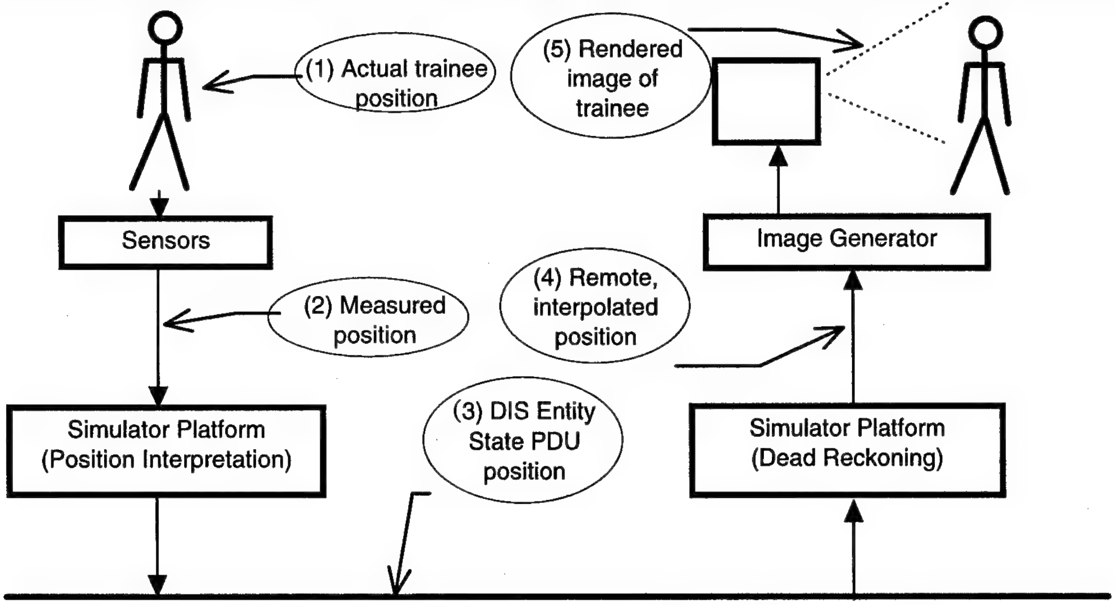

(1) Actual trainee position
(5) Rendered image of trainee
Sensors
Image Generator
(2) Measured position
(4) Remote, interpolated position
Simulator Platform (Position Interpretation)
(3) DIS Entity State PDU position
Simulator Platform (Dead Reckoning)

Figure 2. Five forms of entity state representation in a DIS system

4. 2. Existing DIS coherence limitations

The DIS system only requires new entity state information to be transmitted when an error threshold has been exceeded. Thus DIS allows an error between local and remote position representations in order to reduce network traffic. When the inaccurate dead reckoned state is updated with a new Entity State PDU, the remote entity must either correct the state instantaneously or move it over a smooth, but false, trajectory to the correct state. While this error is unrealistic in any simulation, it may be crucial in a detailed, individual level simulation. Soldiers fighting in close quarters may depend on accurate movements to stay behind covering terrain.

4. 3. Advanced dead reckoning algorithms

In the previous sections we described the need for detailed body position information in the Entity State PDU. A human figure requires many more variables to represent than a typical DIS tank, whose position is typically encoded with a hull location and two articulated parts (turret and gun). For example, the human body has about 200 degrees of freedom [3]. The *Jack* human figure model used in our prototype system has 73 joints [4]. A recent proposal for an Entity State PDU for humans suggested 14 articulated parts [5]. Even more importantly, the normal motion for human limbs is characterized by frequently changing accelerations. Thus for the same number of simulated entities, human figures would easily require more than an order of magnitude more network bandwidth than do tanks.

Under the assumption that joint angle information could not be supported in a large-scale simulation exercise, several advanced dead reckoning schemes have been proposed. In our prototype system, the simulation platform measures head height and applies thresholds to abstract a posture for the DIS representation. In addition, foot pedals are used to control entity speed. At remote platforms, an animation sequence constructed with *Jack* is used to recreate a smooth transition between postures, to create limb movement for a gait appropriate for the entity's speed, and to smooth the transition between gaits.

Even more abstract dead reckoning systems are possible. Several research groups in the computer graphics community have investigated functional, task-level, or goal-directed mechanisms for specifying human figure movement [3] [6] [7] [8] [9]. For example, if an entity's task were to walk along a give line, the animation system could automatically change the remote entity's movement behavior so that the human figure steps over or ducks under obstacles. An intelligent automation algorithm could also be given the goal of moving to a destination location; this dead reckoning model would move the entity around obstacles, turn it to follow corridors, etc. It is not practical to require the dead reckoning algorithm to do too much, however, because it becomes necessary for the simulation platform to sense the trainee's *goals* as well as his position.

4. 4. Coherence problems with advanced dead reckoning

Unfortunately, advanced dead reckoning algorithms can introduce additional correlation errors that are probably not acceptable. Suppose that the simulator could detect a trainee's goal destination and that a remote simulator moved a human figure to that goal. There is no way to guarantee that the rendered figure will follow the same path as the trainee. Small differences in time and space could be crucial to the survival of the trainee. Even the animated posture changes currently in use may result in errors between the trainee's intended position and the position of the rendered figure.

An even more basic problem with intelligent dead reckoning is that it does require that a trainee's intentions can be detected. If a trainee begins to lower his body, at what point does the simulator platform decide that he is going to go to a Kneeling or Prone state? The trainee's simulator platform is in effect sampling his actions when it computes abstract postures; the remote platform, as it generates smooth transitions, is effectively applying a low-pass, smoothing filter to the samples. Delay is inherent in this process. The remote simulator can only reflect the state change after it has already happened.

Clearly, experimentation with the end users will be necessary to determine an adequate compromise between network load, smooth visual displays of moving figures, and coherence between the trainee's state and the remote visual display. We expect the compromise to have these characteristics:

- The trainee's lower body will be partly instrumented, but for the most part will be abstracted locally and recreated remotely. This is reasonable because the trainee is constrained to stay in/on the simulator and cannot actually use his legs to move realistically. The gait would be reconstructed from the upper body position, orientation, and velocity.

- The animation sequences for movement will be based on dynamic models and will [locally] constrain the trainee in terms of acceleration, change of direction, and gait change. Actions that are instantaneous with, say, foot pedal controls will not necessarily take effect instantaneously, even at the local platform.
- Movements that the trainee can control with the position of his body, such as rising and falling, or quick moves within the range of a head position sensor, will be reflected as quickly as possible on the local and remote platforms, even if this causes a jump in position on the remote display.
- The upper body will be more completely instrumented to allow throwing motions, object manipulation, arm signals, head turning, etc. These motions would not be dead reckoned, but would be updated at frequent intervals (e.g. 5-10hz).
- Network bandwidth requirements will be reduced through the use of a human-specific Entity State PDU variant, or data compression, or both.

5. Object representation

The representation of objects in DIS is essentially limited to entities. Having representations for other objects would be useful in general, but is especially needed for individual combatant level simulations. Soldiers typically carry many pieces of equipment, supplies, and weapons that they use in combat. They may carry ammunition for themselves and squad weapons, pieces of squad weapons, grenades, mines, entrenching tools, radios, flak jackets, binoculars, etc. in addition to their own weapons. These objects are worn in various positions, held in various ways and in various combinations, used stowed, expended, dropped, picked up, put in other objects, and given to other soldiers.

If objects are to be present in the world independent of entities, then they require a new type of PDU to indicate their status. DIS 3.0 [10] has introduced the Destructible Entity protocol which could be used for dynamic objects. This protocol is intended for inanimate objects. The object does not itself broadcast any PDU's; rather, entities which act upon the object broadcast "Modification" PDU's when they change the object. Entities can destroy the object. This protocol would be appropriate with the understanding that the entity would not have to transmit a continuous stream of Modification PDU's. The simulator controlling the carrying entity would become the temporary owner of all of the carried objects. We propose that the Modification PDU be extended to allow an entity to indicate that the objects are henceforth attached to itself.

If a soldier (or vehicle, for that matter) is carrying objects, then it should indicate what and where they are. In the spirit of the Destructible Entity protocol, the attached objects could be described once and then not described again in a PDU unless they were modified somehow (expended, dropped, etc.). The initial description would have to include the attachment locations on the entity. Naturally, the attachment positions and the objects themselves would be enumerated in the DIS standard.

6. Weapons fire

Weapons fire is represented in DIS with a Fire PDU followed by a Detonation PDU. Each of these can indicate in a Burst Descriptor that the event contained multiple rounds. However, there is no provision to indicate where each round went. This limitation is acceptable in a simulation with aggregate entities (e.g. fireteams), but not in a detailed simulation with individual soldiers. Each round may be significant. The rounds in a burst will normally scatter and could strike multiple targets or impact in distant locations. At the very least, the Fire PDU should indicate a standard scatter pattern; detonation PDU's should be capable of indicating multiple impact locations from the same fire event.

In addition to impact locations, simulations also need to know munition trajectories (at least roughly) because soldiers hear the rounds passing by even if they don't impact nearby. The trajectories cannot always be computed from impact locations because rounds may leave the terrain database before impacting. Fire PDUs must therefore provide basic trajectory information.

7. Summary and conclusions

In the process of developing the TTES system we have discovered several areas where DIS is deficient for representing individual soldiers.

- The Force ID, Alternate Entity Type, domain and subcategory parts of Entity Type, and posture, primary and secondary weapon status parts of Appearance are not useful for high-fidelity simulation and can be ignored.

- To provide approximate state information for low fidelity simulations, posture and weapon positions can shadow the true configuration information; new fields for low-fidelity location, velocity and orientation would provide values appropriate for use with posture.
- Entity location should refer to the upper torso. Entity state should include detailed upper body limb positions to allow detailed representation of entity actions.
- New fields should be added to the Entity State PDU to describe the position and orientation of the primary weapon.
- A new dead reckoning algorithm for human entities should be introduced for animating gaits based on location (i.e., height) and velocity. Updates to the location of remote entities should be instantaneous rather than smoothed.
- A protocol such as the Destructible Entity protocol should be introduced for representing objects. However, it should be possible for human entities to attach objects to themselves and take ownership of them.
- Fire PDUs should indicate the scatter pattern of multi-round bursts. In addition, they should indicate the initial direction vector of the burst.

With these changes we expect that DIS could support an individual soldier simulation with an adequate level of simulation fidelity, visual realism, and network load.

8. References

[1] IST. *Standard for Distributed Interactive Simulation—Application Protocols, Version 2.0, Fourth Draft (Revised).* IST-CR-94-50. Institute for Simulation and Training, University of Central Florida. 1994.

[2] IST. *Enumeration and Bit-encoded Values for use with IEEE 1278.1-1994, Distributed Interactive Simulation—Application Protocols.* IST-CR-93-46. Institute for Simulation and Training, University of Central Florida. 1993.

[3] Zeltzer, D. "Task-level Graphical Simulation: Abstraction, Representation, and Control." In *Making Them Move.* N. Badler, B. Barsky, and D. Zeltzer, editors. Morgan Kaufmann. 1991.

[4] Granieri, J. "*Jack* and Virtual Reality". In *Quarterly Progress Report No. 51*, N. Badler editor. Department of Computer and Information Science, University of Pennsylvania. 1994.

[5] O'Keefe, J. "Protocol Data Units for the Individual Dismounted Human." To appear in *Proceedings of the Eleventh Workshop on the Standards for the Interoperability of Defense Simulations.* Institute for Simulation and Training, University of Central Florida. 1994.

[6] Drewery, K. and J. Tsotsos. "Goal Directed Animation Using English Motion Commands." In *Proceedings of Graphics Conference '86.* Canadian Information Processing Society. Toronto, Ontario. pp. 131-135. 1986.

[7] Badler, N. *et al.* "Positioning and Animating Figures in a Task Oriented Environment." *The Visual Computer* 1(4), pp. 212-220. 1985.

[8] Calvert, T. "Composition of Realistic Animation Sequences for Multiple Human Figures." In *Making Them Move.* N. Badler, B. Barsky, and D. Zeltzer, editors. Morgan Kaufmann. 1991.

[9] Badler, N., Webber, B., Kalita, J. and Esakov, J. "Animation from Instructions." In *Making Them Move.* N. Badler, B. Barsky, and D. Zeltzer, editors. Morgan Kaufmann. 1991.

[10] IST. *Standard for Distributed Interactive Simulation—Application Protocols, Version 3.0, Working Draft..* IST-CR-94-18. Institute for Simulation and Training, University of Central Florida. 1994.

Session 1F:

Planning and Decision Making I

Automated Battlefield Simulation Command and Control Using Artificial Neural Networks

Ivan J. Jaszlics, Sheila L. Jaszlics and Stewart H. Jones
Pathfinder Systems, Inc.
Lakewood, Colorado 80228

Abstract

Contemporary Distributed Interactive Battle Simulations are becoming increasingly large and complex and therefore, difficult to manage. The success of future projects will depend, in part, on the ability to manage aspects of the command and control of forces in an automated and highly predictable manner. Artificial intelligence in general and Artificial Neural Networks in particular offer attractive mechanisms to automate command and control. This paper describes the Linear Interactive Activation and Competition (LINIAC) Model Artificial Neural Network, a high-speed, object-oriented model, for use in several battle simulations and has demonstrated that this is a feasible application of this technology.

1. Introduction

A fundamental consideration in designing battlefield simulations is that they approach realism as faithfully as possible. One difficulty in simulating battlefield command and control is replicating the decision making which it is based on. The purpose of many simulations is to train a part of its audience to make acceptable decisions. Here it is appropriate to have human operators perform decision making functions. However using humans to make decisions for the Opposing Forces, or friendly adjacent and rear forces may be counter-productive and automated Command and Control may be highly desirable. To provide such automation, many current simulations rely heavily on decision algorithms and rule bases coupled with human roleplayers to emulate the human element. Problems may develop because algorithms and rule bases may not be sufficiently error-free and human roleplayer resources are sometimes difficult to commandeer. Automation using algorithms and rule bases may also lack sufficient flexibility to meet changing scenario requirements without elaborate programming. Artificial Neural Networks (ANNs) offer a cost-effective alternative to algorithms and rule bases for generating or replicating human decision making. ANNs are effective because they are based on examples, rather than hard-coded implementations.

2. The Need for Flexible AI Command and Control

There are some obvious difficulties with using algorithms and rule bases to replicate human decision making. Typically they are hard-coded and are difficult to change if the needs of the simulation change. Recent global military developments, such as the dissolution of the Soviet and Eastern Block forces and the emergence of third world military forces, have changed the command and control requirements for battle simulations. Today simulations must be flexible enough to accommodate a variety of military doctrines and modes of operation, often requiring rapid reconfiguration. If it is necessary to implement algorithms and rule bases using syntactically rigid programming or data-entry languages, their development requires programming technicians to translate the command and control requirements into the appropriate language syntax. The steps required to translate the knowledge of experts into a formalized syntax introduce the possibility of miscommunication and misunderstanding, which can result in program errors. Even when good understanding exists, logic errors can be generated inadvertently. A significant amount of testing is required to detect and remove such errors. Finally rule bases and algorithms usually must account for all possible contingencies when analyzing a

0-8186-6440-1/94 $04.00 © 1994 IEEE

problem so that unaccounted for conditions will not generate unintended results in the simulation. Significant engineering effort is needed to ensure that all reasonable situations are represented in the code.

Using Artificial Neural Nets for automated command and control can avoid many of these limitations. ANNs can be implemented as an object class with standard interface and decision methods. A simulation can then create many decision objects of that class, each with its unique environment consisting of input information and a 'connection matrix' which encapsulates the behavior of a particular ANN. The decision method is a relatively simple mathematical process which is valid for a wide variety of decision applications. A well-designed ANN object can accommodate any compatible decision base and faithfully replicate the cognitive reasoning that it has been trained with.

It is possible to design training methods for ANNs using standard graphical interface techniques that do not require any programming expertise on the part of the trainer, so that knowledge experts can train them directly without having to rely on a technical interpreter. Experts can quickly, often intuitively, learn how to use such interfaces to train ANNs directly in sessions that last only an hour or two. Experts can also define the command and control variables using English words and phrases that make sense both to the trainer and the training audience. When training ANNs, it is not necessary to provide examples for all possible input combinations as it generally is with algorithms and rule bases. A neural net may have thousands to hundreds of thousands of possible input combinations, but a small, representative sample of the total is sufficient for adequate training. ANNs are very good at extrapolating the examples they were trained with to cover other, similar examples. The key to good training, of course, is to include examples which cover the broadest possible range of input conditions.

If a simulation must be able to accommodate multiple scenarios to reflect different military doctrines or modes of operation, then it is possible to train ANNs for each scenario and initialize appropriate ANN objects for the specified scenario with the required behavior when starting the simulation rather than modifying the simulation code. This technique also applies to automated command and control for multiple echelons. The decision structure for several echelons may be similar in that each echelon looks at the same set of conditions and makes equivalent decisions. The only difference may be that each echelon may use different reasoning to arrive at comparable decisions. Therefore it is possible to apply a single decision structure to several echelons, but have each echelon uses a set of ANN objects uniquely trained to reflect its individual reasoning.

Finally it is possible to train neural networks incrementally. If an ANN demonstrates inappropriate behavior within a simulation, it is possible to retrain it quickly. In fact, training through simulation scenarios is a very effective training method.

3. The Linear Interactive Activation and Competition (LINIAC) Model

As part of its research in using ANNs for command and control applications, Pathfinder Systems, Inc. has developed the Linear Interactive Activation and Competition (LINIAC) ANN Model. Figure 1 illustrates how the LINIAC model makes decisions. Each LINIAC ANN consists of an input vector, shown as downward pointing arrows, an output vector, shown as right-pointing arrows, and a connection matrix. The input vector defines a set of input conditions, where each condition can assume one of two or more states. The actual number of conditions and states for a given ANN is arbitrary, but cannot change once the ANN has been trained without requiring retraining. In the LINIAC model, each condition may assume only one state, which is expressed by a 1 value, while all other states for that condition are expressed by a 0 value (although weighted numerical values are also possible). The LINIAC output vector consists of one condition also with an arbitrary number of states. The black dots, shown at the intersection of each horizontal and vertical arrow, represent the neural connections between input and output vector elements and the size of each dot suggests the relative strength or 'weight' of the connection. The weight determines how strongly each input state influences the corresponding state of the output vector. A LINIAC decision is always selected as the output state with the greatest cumulative value. The key to the successful operation of LINIAC is establishing the values of the connection matrix during training so that a given input pattern will always produce the outcome which the trainer has specified.

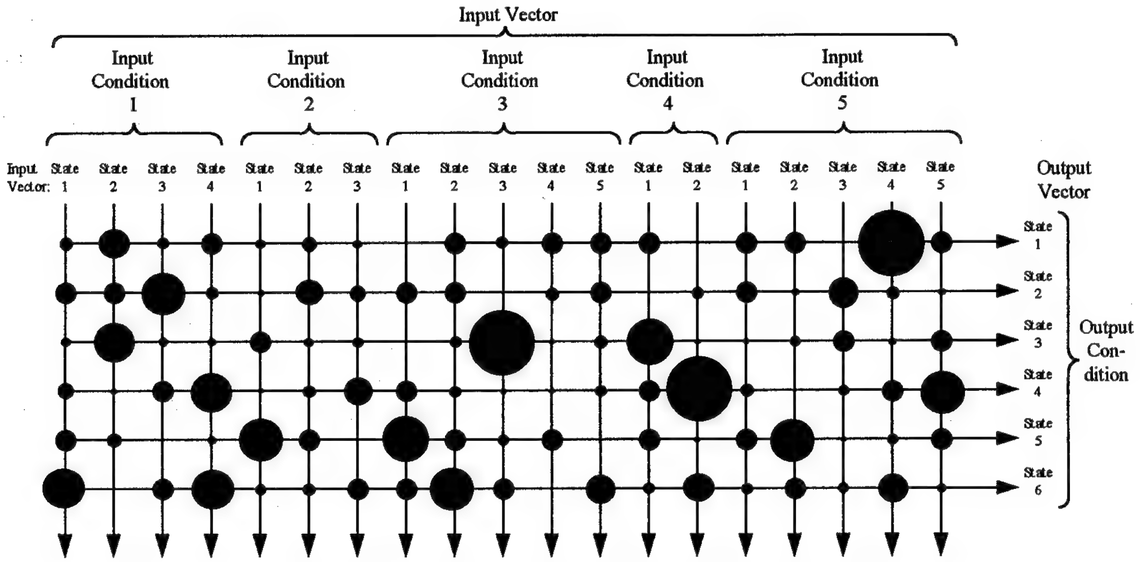


Figure 1: LINIAC Artificial Neural Net Concept

4. Defining and Training LINIAC Neural Nets

To facilitate creating and training LINIAC ANNs, Pathfinder Systems created the Course of Action Planner (COAP) application. COAP is a graphic-based interactive program that provides the ability to define and train neural nets quickly and easily. COAP uses two basic views or windows. The first provides the ability to define and name the conditions and states for the input and output vectors. The second enables an 'expert' to train the ANN and to review and test previous training. A training session consists of entering sets of representative examples by selecting a state for each condition and commanding COAP to learn the behavior specified in those examples.

ANNs must be trained before they can function correctly. Training establishes the 'neural' connection values (weights) between the input and output vector elements of the network. LINIAC neural nets retain their training by saving essential information in an external text file. This output file is useful both for initializing an instance of a neural net in a client object (application) or for review and additional training using COAP. In addition to defining the connection matrix, this external file contains information to enable the neural net engine to map input and output state values to the correct locations in the input and output vectors. Training usually requires a few seconds to several minutes per example to process. The length of time depends primarily on the number of examples already incorporated in the network and how closely a new example replicates a previously learned example. Since thirty to fifty examples may be adequate to provide acceptable output decisions over a wide range of input conditions, a user may be able to train a network within a period of several hours. On rare occasions, it is possible for two training examples to represent patterns which are inconsistent so that the training algorithm cannot resolve the differences (i.e. the back propagation algorithm cannot converge to a solution). If this happens, it is necessary to review the training examples, eliminate the anomalies and retrain the network.

The Review mode enables the trainer to review or test all previous training. Since COAP preserves all training examples in its external text file, it is possible to break training into multiple sessions. Thus the trainer may review previous training in a later session for validation or to avoid redundant input. This also enables a user to retrain or provide additional training for a neural net if the initial training proves to be incorrect or inadequate for the intended application. Another Review mode function is the Performance Test option. This option performs a number of sequential executions of the neural net and displays the average execution time to the user. On an 33 mHz. 80486 PC the average time will vary from a few milliseconds to a few tens of milliseconds, depending on the size of the vectors (and therefore the number of connections). This contrasts favorably with many other current neural net implementations which require substantial computation times. The speed of LINIAC's computation cycle makes it quite attractive for many command and control applications since it can typically perform one to several orders of magnitudes faster than comparable rule-based or algorithmic implementations.

5. LINIAC application to Automated Command and Control: The ROLEPLAYER Model

PSI originally developed the ROLEPLAYER model to demonstrate the feasibility of using ANNs rather than human roleplayers to control portions of a battle simulation. There are many ways to apply neural nets to automated command and control ranging from single neural nets to very complex decision structures composed of layers of neural nets. Although single LINIAC ANNs consider only a limited number of input conditions and make relatively simple decisions, it is possible to group multiple ANNs into a structure which is capable of making much more sophisticated decisions. This is analogous to the way that complex organizational decisions, particularly battle decisions, are typically made. The structure approach is also attractive because it allows the designer to partition complex decisions into simple components which are much easier to understand, design, and train. The success or failure of using LINIAC ANNs for command and control depends heavily upon the validity of the design of the decision model. The value of the LINIAC approach is that simple elements can be designed, redesigned and connected into a structure that accurately represents the decision making process of a real military unit. A workable decision structure will probably be a hybrid of algorithms, rules and neural nets working together. Algorithms are needed to transform simulation data into data types (command and control variables) that are appropriate for input into the ANNs of the decision structure. Of course, not all types of decisions are best implemented through neural nets - when the number of input possibilities and the number of outcomes is small, algorithmic rules should be the better choice. We tend to use neural nets when the possible combinations of the inputs, even if they are not fuzzy, can run into the thousands, or hundreds of thousands.

The training of a battlefield decision ANN is almost trivial if it is performed by a subject matter expert. What is very important is to determine the overall decision structure for an activity represented by an ANN (this can be, for example, a specific human C2 function, such as "armor battalion (BN) S2 - evaluate the current situation"). It is to be expected that a decision structure design will undergo an evolutionary process which will improve its realism. The principal elements of a design include the set of decisions (the ANNs) that the simulation requires at each command and control point, the structure of each decision process (the conditions and states of each ANN), the connections to the simulation data base and the interconnections between the selected ANNs. How individual ANNs are trained is of lesser importance initially, since training or re-training can occur after implementation.

ROLEPLAYER demonstrates the interaction between several friendly (Blue) battalions, controlled by a human operator and several opposing force (Red) battalions controlled primarily by ANNs. The model uses six neural nets: three at the battalion level, two at the company level, and one at the platoon level. Figure 2 shows the structure and interaction of the neural nets at the battalion level. Each net receives a number of input conditions, which are listed above each net box, and produces a single outcome decision value. Each input condition and the output decision are described as a set of states (the states are not shown on these figures). For example, the Enemy Move State condition, which is an input to all three neural nets, can assume one of the States: Marching, Attacking, Halted, Defending and Withdrawing. The ROLEPLAYER model provides state values for these conditions through conventionally-coded algorithms.

The Evaluate Intelligence neural net provides an overall intelligence estimate of the enemy based on observations encoded in its input conditions. This network executes periodically, once every five minutes, to produce a current assessment. It also can respond to events indicating sudden changes in the tactical situation. The net emulates the tactical situation evaluation activity of the Battalion's Intelligence

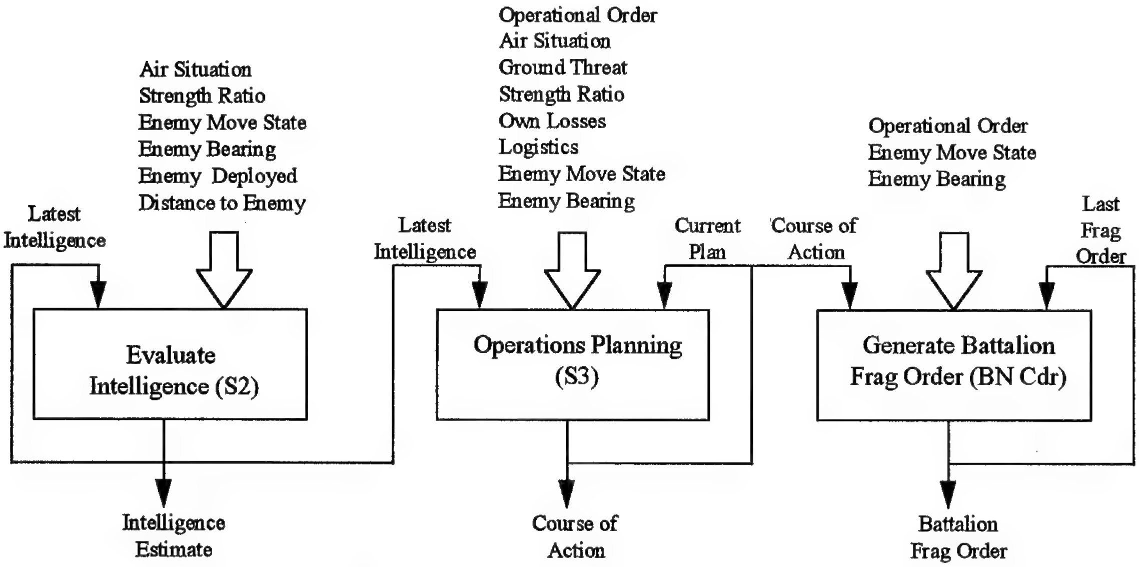


Figure 2: Roleplayer Battalion Command and Control

Officer (S2). The appropriate subject matter expert to train this net would be an actual battalion S2. Note that the intelligence estimate, "Intel Estimate", also has a feedback path into the Evaluate Intelligence neural net and is an input into the Operational Planning neural net. This represents communications from the S2 to the Operations Officer (S3). The feedback of earlier recommendations into the net itself represent the fact that situation evaluation is not likely to change immediately, without considering previous estimates. Conditions which are output from one neural net and input into another net must have state sets which are identical in the number of states and their order.

The Battalion Operations Planning ANN receives the Operational Order and Fragmentary Operational Orders (Frag Orders) from higher headquarters. It considers this order along with other conditions including the Latest Intelligence Estimate output from the Evaluate Intelligence neural net and determines what course of action to take at the battalion level. This net executes periodically, once every five minutes or in response critical tactical condition changes, to produce a current course of action. Depending on the input conditions and the encoded training, the ANN will recommend to continue carrying out the current mission, or to follow another, more appropriate, course of action. It essentially emulates the immediate operations re-planning and course of action determination activities of the Battalion Operations Officer (S3). The optimal subject matter expert to train this net would be an actual S3 for Blue forces, or an Intelligence officer familiar with enemy doctrine, equipment, and tactics for the Opposing Force (OPFOR). This net also considers its last recommendation as one of the inputs, and provides its recommendation to the Battalion Frag Order net.

The battalion Fragmentary Operational Order (Frag Order) neural network is responsible for deciding what Frag Orders the battalion will send to the company commanders under its command. It executes periodically, once every five minutes, or in response to tactical emergencies, to produce a new Frag Order. It considers the Course of Action decision, produced by the battalion Operations Planning ANN, and also its previously issued Frag Order as a feedback from its previous execution. The most common outcome from this ANN is a "Continue" decision, which means that there is no change to the order which each company is carrying out. Again, the decision it actually makes depends on how the network was trained. You may observe that the input conditions for each of these neural nets appear to be arbitrary. What is included as input conditions to a neural net is a decision which the simulation designer must make jointly with military subject area experts.

ROLEPLAYER uses similar Frag Order decision ANNs at the company and platoon levels to provide equivalent Frag Order command and control decisions at those levels. The ANN at the company level executes once every two minutes, or in response to tactical emergencies, and at the platoon level, once every minute, or in response to tactical emergencies. The design of these nets is similar to the design of the battalion Frag Order net, but each net must be trained independently to behave just as the company or platoon commander would.

ROLEPLAYER also processes a Fire Support Request ANN at the company level which requests external fire support when battle conditions warrant it. If the ANN decides to request fire support, ROLEPLAYER passes the request to a corresponding Battalion Fire Support Coordination net at the battalion level. The FSC net will, depending on available assets and on the Battalion level evaluation of the tactical situation, either grant or disapprove the request. If the request is granted, the FSC net also decides the allocation of appropriate assets (indirect fire or air support) and sets the execution of the support activity into motion. Time delays between approval of support requests and actual support are due to factors which may be directly a part of the simulation (such as the movement of aircraft), and factors indirectly included in the simulation (additional C3 delays, time required to shift fires, take-off time of ready aircraft, etc.).

6. Distributed Interactive Simulation (DIS) Applications

The structure of LINIAC neural net inputs and outputs are well suited for implementing them in a DIS environment. Information can be passed to a neural net in he form of ***Condition : State*** value pairs or ***Net Name : Condition : State*** value triplets. This can easily be implemented in a Protocol Data Unit (PDU) such as the Signal PDU in the IEEE standard. Values can be expressed as character strings, enumeration types or appropriate integer values. The Net name may not be required if the command and control application which utilizes the neural nets can determine which net(s) to execute from the context of the incoming data.

Another client - server approach has been implemented by applying LINIAC to the U. S. Army's EAGLE simulation using an Object Request Broker (ORB or CORBA) to execute a decision structure, which itself is processed as an object. The Object Broker passes a number of decision variables to the decision object through an ORB call. The decision object may exist on a different platform than the simulation, and consists of a network of ANNs. When it receives an ORB call, an interface process allocates the included decision variables to the correct ANN input vectors, executes the ANN decision making functions and returns the appropriate outcomes to the client through the ORB return.

7. Conclusion

This paper has presented a practical approach for using artificial neural nets to perform automated decision making in the context of combat simulations. Neural nets can be much easier to design and implement than comparable algorithms or rule bases. A single neural net engine can function as a server for an arbitrary number of neural nets. The 'code' required to execute a neural net can be encapsulated in an external data file, including both the connection matrix and the condition/state definitions for the input and output vectors. Because of this external encoding, the behavior of a client can be modified simply by substituting a differently trained input without changing source code. This greatly reduces the amount of time and the expense required to design, implement and maintain decision logic/code for automated forces. In simulations this provides essential flexibility because the behavior of automated forces may need to change to reflect different scenarios. This also makes it possible to replace neural nets whose initial training may contain deficiencies.

Because it is possible to train neural nets using a relatively simple graphic interface, it is possible to have 'experts' train them quickly and directly without requiring intermediate technical personnel who may inadvertently introduce personal biases into the decision base. This user interface also provides the capability to review the training and behavior of a neural net and thus provides a first level validation for the behavior of the net. Neural nets reflect their training examples very faithfully and avoid unnecessary errors caused by coding anomalies. They are also very good at exhibiting behavior for which they have no discrete training by extrapolating learned examples to cover those conditions, which greatly reduces the time that experts must spend in training them.

The LINIAC neural net implementation possesses a very fast execution speed, and even a structure of multiple neural nets operating sequentially to produce a single decision may easily out-perform a comparable algorithmic or rule-based implementation. Finally, PSI has verified their performance, reliability and accuracy in several demonstration projects.

"GAME COMMANDER"—APPLYING AN ARCHITECTURE OF GAME THEORY AND TREE LOOKAHEAD TO THE COMMAND AND CONTROL PROCESS

A.Katz
University of Alabama
Tuscaloosa, AL 35487-0280
akatz@ua1vm.ua.edu

B. Butler
Loral Advanced Distributed Simulation
Orlando, Fl 32826
butler@orlando.loral.com

Abstract

This paper describes an architecture for emulation of those portions of the higher-echelon command and control task which deal with planning and evaluation of courses of action. (COA). The architecture described utilizes the technique of tree lookahead with game theory. Tree lookahead is a technique for computing optimal decisions. In chess it is now the preferred method in real time. It is precisely the chess application which motivates its adoption into an overall command and control architecture. Chess is the prototypical war game and the ways that chess players develop their strategies are not unlike the techniques used by commanders to reason about, plan, and evaluate courses of action to pursue in a combat situation. The paper examines implementation issues associated with this technique. It reviews some past work on developing this technique for controlling rotary-wing aircraft. It establishes how this architecture could be embedded in a virtual simulation Computer Generated Forces host (e.g. "Modular Semi-Automated Forces" or ModSAF) that would assume other command and control functions and thus provide a broader context for command and control.

1. Introduction

This paper describes an architecture for emulation of those portions of the higher-echelon command and control task which involving planning and evaluation of courses of action (COA). The architecture utilizes the technique of tree lookahead with game theory. Tree lookahead is a technique for computing optimal decisions. It is native to game theory where it serves as a tool for both theory and practice. In chess it is now the preferred method in real time. It is precisely the chess application which motivates its adoption into an overall command and control architecture. Chess is the prototypical war game and the ways that chess players develop their strategies are not unlike the techniques used by commanders to reason about and plan the course of action to pursue in a combat situation.

2. Emulation of command and control

Command and control, as defined by the US Army, encompasses a broad range of tasking which can be decomposed into four subtasks: (1) Acquire and communicate information and maintain status, (2) Assess situation, (3) Determine actions, and (4) Direct and lead subordinate forces [17]. The lookahead architecture provides solutions for the emulation of portions of the subtasks of "Assess situation" and "Determine Actions".

Most direct is the emulation of the subtask of Determine actions. Of its five component sub-subtasks, the lookahead architecture can be applied to these four: develop courses of action, analyze courses of action, compare courses of action, and select or modify course of action.

Emulation of the Assess situation subtask is less direct. Reference [17] describes that this subtask has the commander "continuously evaluate information received...to decide whether different actions are required from the most recent orders issued". In the paradigm of Army command and control, this subtask is separate from that of Determine Actions. Its component sub-subtasks are: (1) Review current situation, and (2) Decide on need for action or change. Note that in practicality, the decision not to determine a new action, to instead stay the course, is an explicit decision. This practical view captures the spirit of the lookahead. The lookahead will generate a decision at every decision cycle. The outcome of the process may be the continuance of the previous decision. The lookahead executes the second sub-subtask (Decide on need for action or change) by its continual review of the current course of action. It also executes much of the first sub-subtask, with the exception of the information management portions of that sub-subtask.

In order to perform the broader task of command and control, we advocate that the lookahead architecture be embedded in a virtual Computer Generated Forces (CGF) simulation capable of performing the full range of command and control tasking. The paper describes how this could be done with the "Modular Semi-Automated Forces" (or ModSAF) CGF.

3. The mathematical theory of games

The problem of decision making by agents with conflicting goals is addressed by the mathematical theory of games. The groundwork of game theory was laid by Von Neumann and others in the first half of the century [14]. This classical work addressed discrete games.

0-8186-6440-1/94 $04.00 © 1994 IEEE

Differential games that deal with continuously varying systems were added later. In computer applications everything is discrete. For this reason we will accept that continuous games can be approached by a limiting sequence of discrete games, and that a fine enough discrete game yields an acceptable representation of the continuous game.

Discrete games come in two varieties: ones with simultaneous moves and ones with consecutive moves by the players. Intuitively, it would appear that either kind, if fine enough, could represent a differential game. However, the two varieties exhibit vastly different properties. It is only the games with consecutive moves that admit deterministic optimal solutions, as do differential games. For this reason we select the discrete game with staggered decisions as our basic model of reality.

We also focus our attention on zero sum games, where the gain of any party must exactly equal losses by the others. The assumption of zero sum is inadequate for most economic and political contexts. A voluntarily-consummated market transaction could benefit both parties because of the differing utility functions each brings to the exchange. Yet in war games it is nearly true. Objections may be raised to this assumption (deception, pyrrhic victories, mis-perceptions), but they can be handled by the mechanics of the lookahead. Consider deception. A commander may be tasked with attacking a stronghold, with consequent heavy losses, in order to deceive the enemy. The lookahead framework can accommodate this deception by building in appropriate components of the heuristic evaluation of the "goodness" of outcomes represented by terminal (or "leaf") nodes. Pyrrhic victories are possible outcomes in military conflict, but are never a commander's goal. Even in the case of deception, a commander would *want* to prevail with minimum losses, but understands that conditions will not permit. It is therefore appropriate to emulate the commander's thinking in a zero-sum context. Perceptions may also enter in and color the commander's evaluation of the tactical situation. Both of the players may determine that they have lost (one of them wrongly deciding so) and act accordingly. This could be handled by separate maintenance of perceptual truth versus "true truth", and feeding perceptual truth to the lookahead engine.

Within the framework of assumptions noted above, classical game theory offers some powerful insights and tools. It is proved that zero sum, staggered-decision games possess optimal solutions. The solution may, in principle, be found by constructing a *game tree*. It may, in principle, be stated as a *strategy*.

A strategy is a set of rules. The rule based approach that is prevalent in contemporary artificial intelligence (AI) falls within the realm of approximate strategies. There is a wide class of problems that lends itself to this method. The AI community has been busily attacking these problems over the last two decades. But some problems are too complex and too rich. One that stands out is the game of chess. In that area, tree lookahead in real time is the method of choice, which has been remarkably successful [13].

The game tree and the strategy are equivalent abstract tools, which, if expanded exhaustively, embody the solution of the game. But the game tree and the strategy are also practical tools for constructing approximate solutions. The nature and richness of the game determine which tool is more appropriate for practical use.

We suggest that the wargame problem at the level of the platoon, company, and battalion commanders falls outside of the class that yields to the rule-based approach. It is rather more like that stylized and abstracted wargame chess, and like chess, its solution must incorporate lookahead in real time.

4. The lookahead technique applied to the decision-making commander

4.1 Tree Expansion

Figure 1 illustrates the game tree associated with the lookahead process. Without loss of generality, assume that the game tree describes the decision-making process for the Blue Commander; his opponent being the Red Commander. Each circle in the diagram (called a "node") represents a complete battle situation—characterized by position and disposition of both enemy and friendly forces. The topmost node (at "ply-level" 0) represents the current battle condition that calls for a decision from the Blue Commander. To each potential decision alternative (represented in the diagram by the tokens "Attack", "Defend", and "Hold") there corresponds a branch emanating from the node. Each branch represents the unfolding battle histories resulting from implementation of the selected decision.

Nodes immediately below the top-most node are at ply-level 1 and represent the potential battle outcomes one ply-interval later. Further development from ply-level 1 proceeds by alternating the turn of the decision-maker to the Red Commander. All possible decisions by the Red commander are considered for each node, and corresponding branches are constructed. These branches are continued to ply-level 3, where the Blue commander's decisions are again used for branching. The tree is recursively developed in this fashion as deeply as computing and time resources permit. Nodes correspond to specific points in time. Branches represent the unfolding development that results in the next-lower node outcome when starting from the decision-making commander's decision. The direction of time in the diagram is down.

When the game tree cannot be exhaustively developed, the leaf nodes do not represent obvious win-lose-draw outcomes, but rather intermediate and indeterminate outcomes. In this situation (which is normal given resource constraints) the leaf nodes need be assessed by a heuristic score which can conveniently be bounded between -1 and 1; 1 being best for Blue and worst for Red. The heuristic score is propagated up the branches to become a score which valuates the entire decision-making opportunity. This propagation is done by simulating the action. Ideally this might be done with full virtual modeling of the platforms involved. Considerations of computer resources will usually restrict these calculations to determining motion and attrition on an aggregate level.

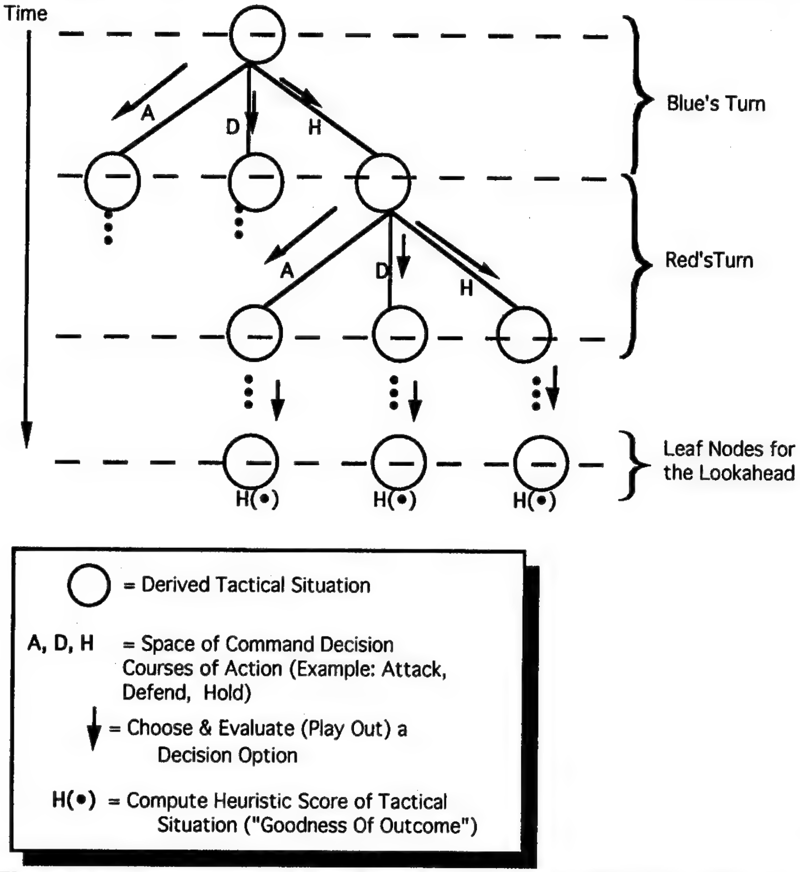

Time
A
D
H
Blue's Turn
A
D
H
Red'sTurn
Leaf Nodes for the Lookahead
H(•)
H(•)
H(•)
= Derived Tactical Situation
A, D, H = Space of Command Decision Courses of Action (Example: Attack, Defend, Hold)
= Choose & Evaluate (Play Out) a Decision Option
H(•) = Compute Heuristic Score of Tactical Situation ("Goodness Of Outcome")

Figure 1: Illustration of the Tree Lookahead with Game Theory Technique

Note, however, that the attrition equations, introduced in [7] and expanded upon for this application, are not subject to the criticisms that Lanchester's equations usually raise [6]. This is because the equations are embedded in the overall decision-making process of lookahead tree expansion which can adequately respond to changing positions. The equations are applied only for the short duration of ply-intervals. Even if some effects of attrition, such as retreat and regrouping are not represented inside the ply-intervals, an adequate spectrum of decisions is available at the nodes to compensate.

As scores are propagated back up the tree, several branches meet at each node bringing with them varying scores. If it is a Blue commander's decision node, the branch that scores highest is selected. If it is the Red commander's, the branch with the lowest score is selected. The score of the selected branch becomes the node score.

The propagation is continued all the way up to the root node. The score reaching the root becomes the overall assessment of the situation. The decision engine returns the command associated with the selected branch.

It has been the experience in chess that the assessment refined by several levels of tree lookahead is greatly improved over the original heuristic assessment, and not sensitive to the details of the heuristic scoring.

Note that the decision made allows for Red's most effective counter moves. Using the classical "min-max" process from game theory, Blue embarks on a course of action leading to the "best achievable outcome"—one that Red cannot prevent.

Even though the selected branches of the tree represent the unfolding of long range plans, only the initial phases of these plans are put into operation--unless Red were to behave at each decision point exactly as predicted. Once a decision is reached, handed down, and execution begun, the whole process starts over again. Actual situations rather than preplanned moves drive the decision process. Mistakes by Red are not anticipated (because of the min-max paradigm), but are exploited as they occur.

4.2. Score assessment

Two kinds of scores are used in the lookahead process. We appeal to the chess analogy to help motivate understanding of the distinction.

First we look at the branch scores. Both sides start out with their best possible scores which are normalized at a value of one. In the chess analogy, the score may represent remaining strength. At the start of the game, no losses have occurred, hence the score normalized to one means a full complement of pieces. In the game commander application, a score of one means all the forces remain with which one started the decision cycle. As the battle progresses, losses are occurred and can be measured directly. In the tabulation of losses, we see clearly the distinction between the chess analogy, which is a truly discrete game, and the Game Commander application which is a continuous game, approximated discretely. In chess, losses occur at the nodes when pieces are taken. Hence, branches of the game tree don't represent intermediate states of the game. Instead they represent logical connections between nodes. In the Game Commander application, losses occur at the branches between the nodes because the nodes represent game states which are discrete instances of time. Actual combat losses must occur during the branches which represent time intervals connecting the nodes. In either case, losses can be measured exactly at the nodes. In the case of chess, the score is non-increasing with time to represent the loss of total strength as pieces are taken. The same hold true in the game commander application.

At leaf nodes, the scoring changes. Terminal scores are estimates of the situation made necessary because the game will not be played to any deeper extent. In the chess analogy, these scores are necessarily heuristic and account for relative positional advantage between the players. There is no need to account for relative strength advantage, because loss of strength has already been accounted for in playing down to the terminal ply-level. In the Game Commander, position must also be accounted for as well as other intangibles such as morale, fatigue, supply state, etc.

5. Related Work—Aerial Combat

An adaptation of the tree lookahead with game theory technique has shown promise in the cognitive task of controlling helicopters in air to air combat close to terrain.

References [10], [11], and [12] describe work done on a project called Intelligent Player (IP) which was intended to exploit the lookahead technique. By 1990 a prototype of IP was flying against a manned Apache simulator. More advanced IPs ran off-line, conducting deeper searches without the burden of real-time. References [7] and [15] describe the progress of the work since that time. In particular, [15] addresses some recent advances in optimizing Intelligent Player's real-time performance.

A different application of the lookahead technique is Automan, a program created by Dr. Fred Austin and others at the Grumman Research Center and installed at NASA AMES (see [1] and [2]). Automan is the only existing lookahead that copes with hilly terrain in real time. Automan also differs significantly from IP in that it operates by allowing simultaneous turns.

All the prototypes to date were severely limited in depth of the lookahead—one ply for IP, two plies for Automan. Off line studies indicated a dramatic improvement in intelligence at the three-ply level —the first point at which a plan can be formulated. Reference [15] concludes that four-ply tree searches are now feasible for the aerial combat application with PC or workstation equipment. The time for useful lookahead, in the aerial combat arena, may be at hand.

It is natural to consider how the success of the lookahead in command and control at the platform level will translate to upper-echelon command and control. Every command level fits into an larger hierarchical structure. As does a real commander, Game Commander must be able to take orders or "missions" from echelons above, formulate a plan, execute the plan by producing orders for subordinate forces, and monitor plan execution by receiving and interpreting the status flowing into the command post. IP operates in this same context. It has a "mission" to fly, fight, and survive. IP formulates a plan as a sequence of aerial combat maneuvers. IP executes this plan by giving commands to the flight model and weapons systems. IP monitors plan execution by continually monitoring tactical state. In both applications, the lookahead decision-making engine does not produce direct control inputs, but rather a discrete decision choice that must be adapted through interpreters. In both applications, the lookahead analyzes a simplified mathematical model of the combat situation and controls through selection of a range of discrete control choices which are sequenced together to formulate a plan.

The differences in the two problems are largely in scale. Game Commander has a wider range of choices in plan formulation than that facing IP. Yet IP must respond to a more swiftly changing situation. In this trade of decision-space range for reaction time, given the real-time viability of IP, we believe a real-time implementation of Game Commander is also viable. See [3] for additional reference of a real-time model of platform-level command and control The prototype FFCS mentioned in this paper also performs in the domain of air warfare.

6. The Turing test

Will command decisions based on tree lookahead in real time be indistinguishable from ones made by a human commander? The following remarks may be in order:

1. The problem with most current systems is that they appear dumb and inflexible in comparison with human decision makers.
2. Once the human level is reached and exceeded, it should be relatively easy to degrade the tree-based decision maker by limiting the depth and breadth of his lookahead. This has been demonstrated by the computer chess community, who offer grand master level players degradable to the level required to instruct and entertain their amateur owners.
3. High level commanders, like chess players, actually formulate their decisions by considering chains of moves and counter moves. This is reflected in the literature [16] and in some interactive planning tools [5]. Intuitively, this lookahead process is continued forward in time until either the range of possible outcomes is too broad to accommodate, or the estimation of likelihoods becomes too degraded due to the depth of the search. In either case lookahead ceases and the space of outcomes must be evaluated as to desirability and to probability of occurrence. This same process holds true for chess grand masters. In the chess analogy, the

terminal states must be evaluated heuristically, and one can say that pattern matching is the obvious evaluation technique. So, like chess grand masters, commanders employ the technique of lookahead to formulate possible courses of action, and the technique of pattern matching (among others) to evaluate the possible courses of action.

4. The semi-heuristic scores of leaf nodes are open to review and refinement by military experts.
5. The whole decision process is open for validation by running it off line against benchmark situations and by producing sample strategies by repeated runs (see Section 7).
6. If a decision engine better than human emerges, it should prove a valuable spin-off product.
7. Lookahead decision making can accommodate the modeling of decision making under conditions of imperfect information. Each decision is made on the basis of the decision-makers's *perception* of the situation, and on his perception of the knowledge and goals of the opponent.

7. Adaptation and learning

Tree lookahead is a powerful tool for analysis and learning. In a classical study [1] the Grumman research team showed how tree lookahead, run off-line, can be used to formulate an optimal strategy. Starting with a strategy of random decisions, the tree lookahead was run repeatedly with the decisions used in winning encounters reinforced, and the ones figuring in losing engagements suppressed. This process, applied to air combat, converged to a strategy that closely matched the one used by experienced pilots.

In this way the lookahead technique invites review, critique, and validation. Matching against human performance is possible even though it is not assumed that the human experts can always articulate their expertise in transferable form.

Many variations are possible in which spare compute time is used to review past engagements. In this way strategies can be fine tuned for a particular adversary, rather than the optimal adversary that the pure lookahead assumes. The reviews can lead to pruning rules that eliminate enemy responses which are contrary to enemy doctrine and practice as they manifest themselves in the building experience of the decision maker.

Pruning rules are subject to off line review by military experts. It is also possible to inject such rules as a-priori assumptions.

8. Implementation issues

The major obstacle that any scheme of tree lookahead in real time must overcome is computer throughput. The volume of different options that must be explored grows exponentially with the depth of search. The following remarks are pertinent.

1. Granularity. The computational burden is alleviated by limiting the range of discrete choices of a command decision and by increasing the time interval by which the command must be reviewed and a new decision reached. In the case of the command forces, this is natural. Commanders normally formulate discrete decisions (e.g., ``Pass north of hill G7 rather than south"); they often have minutes or even hours to formulate a major decision; once put into action, it takes minutes or hours for the situation to change appreciably. All this tends to indicate that the problem of discretization will be less severe in the context of the command forces than it was in previous work (see Section 5). Still, the proper level of granularity must be defined. Modulation of granularity is a natural degradation mechanism for producing more or less capable commanders. To emulate human behaviors it must be confirmed that a level of granularity exists which fits a human commander.
2. Force modeling. A tree lookahead decision engine must be able to exercise the forces involved through many tentative moves as it develops the game tree. This must be done faster than real time. The normal ModSAF-style virtual simulation representation of such forces will, most likely, prove to be too elaborate and will not be able to execute fast enough in the service of even a single decision maker. Different decision makers will want to manipulate the same forces at the same time for different tentative moves. For this reason, the decision engine must come with its own simplified model of the forces being commanded, capable of executing the many tentative moves much faster than real time. (See the discussion of attrition computation in Section 4.)
3. All other measures for conserving compute power notwithstanding, it will be necessary to address the hardware and software architecture necessary to optimize the computer resources involved. Optimization of the supporting software architecture for the lookahead has been addressed in [15].

9. Integration into a command architecture

The lookahead technique must be integrated into a comprehensive architecture for command and control. This paper sketches some of the problems that must be solved to accomplish this integration. A notional layout of the comprehensive architecture is illustrated in Figure 2. It is built around the ModSAF Computer Generated Forces system and utilizes the messages of the Command and Control Simulation Interface Language (CCSIL), developed by the Advanced Research Projects Agency (ARPA) for communication between CGF systems [4]. The architecture embodies explicit representation of commander perceptions as input for the decision-making cycle.

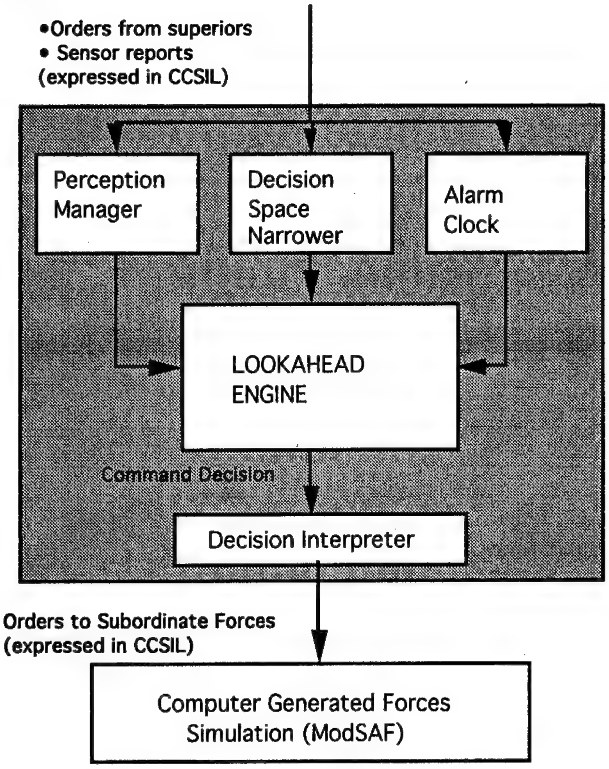


Figure 2: Illustration of the Command and Control Architecture

CCSIL is still nascent and does not yet provide the functionality we seek. It is the idea behind CCSIL that we seek to exploit—a standard language for communication of command and control information (orders and reports) among software commanders in a form that is optimized for data processing. This form of standardization will provide interoperability between the lookahead engine and other CGF implementations which could substitute in the ModSAF role.

Inside the gray box of the figure is where the cognitive processing is concentrated. The lookahead engine, the center-piece of the architecture, continually cycles to assess the tactical situation and to produce the appropriate orders. The orders that issue from it belong to that discrete range of command options available to the command emulator. These command options must be parameterized by the specifics of the current situation and then translated into operational orders (eventually into CCSIL format) which can be executed by the CGF.

Feeding into the lookahead are the orders and reports which shape the environment. We can interpose several kinds of functions between the lookahead and its input in order to tailor lookahead operation.

To add realism to the output decisions of the lookahead by having it decide from perceptions rather than ground truth, one can interpose a Perception Manager between the reports coming in and the report information going to the lookahead. The Perception Manager would maintain an explicit model of the nature, degree, and sources of misinformation on the battlefield and would perturb the inputs to the lookahead to simulate this misinformation.

To optimize the operation of the lookahead in meeting real-time constraints, one can interpose a "Decision Space Narrower" to constrain the development of the decision tree both in depth and in branching. The Decision Space Narrower would examine the status and orders inputs coming into the decision cycle. Firing heuristically-derived rules, it would block expansion of those parts of the decision space that show little promise. These decision space constraints would be conveyed to the lookahead engine in terms of search limits in depth and in prohibition of expansion of certain branches of the lookahead tree.

An alarm clock can also be added to the cognitive architecture in order to cut short an overly lengthy search of the decision space. The alarm clock can be set by a heuristic assessment of the tactical situation to decide how long Game Commander can wait before it must produce a decision—i.e. what degree of urgency exists. The lookahead tree can then be expanded as to most promising branches and depths first. When the alarm clock goes off, tree expansion is cut short and the developed decision options are scored.

The ModSAF CGF provides the simulation action on the battlefield. The orders produced by the lookahead are fed into ModSAF to produce plan execution.

Bibliography

1. Austin, F., G. Carbone, M. Falco, and H. Hinz, "Automated Maneuvering Decisions for Air to Air Combat", Grumman Report RE-742, November 1987.

2. Austin, F., G. Carbone, M. Falco, H. Hinz, and M. Lewis, "Game Theory for Automated Maneuvering During Air to Air Combat", *Journal of Guidance, Control, and Dynamics*, Vol. 13, No. 6, 1990, pp. 1143-1149.

3. Czigler, M., S. Downes-Martin, and D. Panagos, "Fast Futures Contingency Simulation: A 'What If' Tool for Exploring Alternative Plans", Military, Government and Aerospace Simulation MultiConference, Society for Computer Simulation, April, 1994.

4. Dahman, J. S., "Command Forces (CFOR) Program", presented at the 4th Conference on Computer Generated Forces and Behavioral Representation, Orlando, May 1994.

5. Downes-Martin, S., H. Deutsch, and G. Abrett, "Managing Uncertainty in Time Critical Plan Evaluation," International Journal in Man-Machine Studies, Vol 36, pp 337-356 (1992).

6. Epstein, J. M. , *The Calculus of Conventional War*, The Brookings Institution, Washington 1985.

7. Katz, A., "*Intelligent Player*—First Principle Foundations", Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation, Institute for Simulation and Training, Orlando, Florida, March 1993, pp 329-334.

8. Katz, A., "OBD: A Truth Maintaining Inference Engine implemented in Ada", *Artificial Intelligence and Simulation*,, p188, Society for Computer Simulation (1988).

9. Katz, A., "Tree Lookahead in Air Combat", to be published in the Journal of Aircraft (Aug 94).

10. Katz, A. and B. Butler, "A Flight Model for Unmanned Simulated Helicopters", Journal of Aircraft **29**, No. 4, July-August 1992, p521.

11. Katz, A., B. Butler, and D. Allen, "A Computer Generated Helicopter for Air to Air Combat", AIAA Simulation Technologies Conference, New Orleans, Aug 1991, p 82.

12. Katz, A. and A. Ross, "One on one Helicopter Combat Simulated by Chess Type Lookahead", AIAA Flight Simulation Technologies, Boston, Aug 1989, p 357. Also published in the Journal of Aircraft **28**, no. 2, p158 (1991).

13. Levy, D. N. L., *The Chess Computer Handbook*, Batsford, London, 1984.

14. Neuman, J., and O. Morgenstern, *Theory of Games and Economic Behavior*, Princeton: Princeton University Press, 1944.

15. Schaper, G. A., "Lookahead Limits of Intelligent Player", Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representations, Orlando May 1994, pp 401-410.

16. M. Shubik, "Game Theory—The Language of Strategy," *Mathematics of Conflict* (editor Shubik), North Holland, Amsterdam 1983.

17. US Army, "Blueprint of the Battlefield", Army Training and Doctrine Command, TRADOC PAM 11-9, 1990.

Incorporating Simulation-Based Models into Planning Systems

Jin Joo Lee and Paul A. Fishwick
Department of Computer and Information Sciences
University of Florida
Gainesville, FL 32611
jl1@cis.ufl.edu, fishwick@cis.ufl.edu

Abstract

General-purpose planners have been proposed but few have shown to work effectively and efficiently enough for many domains to be really called general-purpose. A general-purpose planner that uses a single methodology is often too restrictive and therefore cannot plan effectively for all domains. As planning problems become more complicated, having multiagents of different types in dynamic environments, evaluating candidate plans and choosing the best plan becomes prohibitively complex if not impossible within a single methodology. To overcome this problem, we propose simulation-based planning where simulation is used to evaluate the candidate plans. By allowing appropriate simulation model types to accurately express each type of agent in the domain, the task of measuring the success and effects of each candidate plans is simplified and the resulting evaluation will be more accurate since plans are simulated using dynamic models. We describe an application along with the implementation of simulation-based planning in the domain of Mission Planning. Possible future experiments related to Soar are also discussed.[1] **[Key Words: Autonomy, Mission Planning, Computer Generated Forces, Multimodeling]**

1 Introduction

General-purpose planners [12, 15, 14, 13] claim to solve planning problems for many different domains. They may, in fact, be able to solve all the different planning problems but the problem is that they may not be able to solve it in an efficient and timely manner. Since many of these general-purpose planners use a single methodology, they are usually restricted in their ways of representation and reasoning. Therefore, they do not have the flexibility to adequately express and reason about all the different domains in an efficient way. This is not a problem when the planner does not have to plan, interact and execute at the same time. However, for any planner that has to work in an interactive environment such as the Distributed Interactive simulation Environments, a planner must not only plan and react at the same time but it must do so in a time period that is at least close to real-time.

Our solution is to use simulation-based planning which uses simulations to evaluate different candidate plans after they have been generated by the planning system. A typical way to view the planning process is to divide it into three steps. The first step is plan generation where several plans, if possible, are generated that are likely to be good candidates. Second, the set of candidate plans are evaluated by performing a temporal projection into the future in virtual time and accessing the results prior to the execution. The results are then compared and a plan is chosen for execution in the final step. If appropriate models can be used that best captures the behavior of actions and reactions of each agent, the evaluation of plans will be more accurate allowing better selection of the best plan.

Multimodeling [5, 4, 6, 8] will be used to model processes and agents at multiple abstraction levels. Related work [9] suggests the use of a coordinated set of methods, each method having different scope and performance. Some experimental implementations of this approach has been done on Soar [11].

In section 2, the general concept of Simulation-based planning is described. We discuss the mission planning problem as an application in Section 3. Then

[1] ISBN 0-8186-6440-1. Copyright(c) 1994 IEEE. All rights reserved.

0-8186-6440-1/94 $04.00 © 1994 IEEE

in section 6, we propose the design of planned future experiments and finally some conclusions in section 7.

2 Simulation-based Planning

Fishwick [6] defines simulation as " the discipline of designing a model of an actual or theoretical physical system, executing the model on a digital computer, and analyzing the execution output". In the AI planning literature, Dean [3] states that the idea of using model to formulate sequences of actions is central to planning and given a sequence of actions, a robot can use the model to simulate the future as it would occur if the actions were carried out. So simulation provides the robot with information which can be used to suggest modifications or to compare the proposed sequence with alternative sequences. Thus, simulation-based planning integrates these two ideas. Simulation has always been used within planning, although in a very abstract way, using operators and rules for example. The idea is to use more detailed simulation models originally built for simulation purposes in place of the highly abstract rule-based models. Therefore, once simulation models are built for a system, simulation can be used as a tool to provide the system with information useful for evaluating its hypothesis which are a set of generated plans.

3 Application: Mission Planning for Ground Combat

The Distributed Interactive Simulation Environment provides the ability to create large virtual worlds by linking individual simulators, allowing them to interact in real-time. And the Department of Defense has used these capabilities to revolutionalize the way the military train their forces, prepare for a combat and plan and rehearse operational missions. Even before the DIS revolution, the military has used the simulation-based approach to planning in the form of conflict simulation (or wargames). Conflict simulations of the constructive type involve aggregate simulations using discrete time (turns) and space (terrain). During a planning phase, a commander would perform "what if" scenarios by setting up a course of action on a hexagonally tiled map of the terrain. Engagements, instead of being fought with individual entities, are abstracted using a stochastic method in the form of a combat results table (CRT) or through Lanchester equations for force attrition. Related work by Czigler et al. [1] demonstrates the usefulness of simulation as a decision tool in military planning.

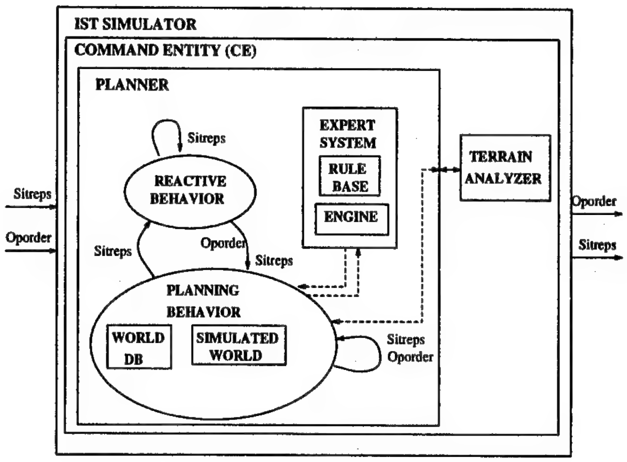


Figure 1: Planner Architecture

Through a sample application in the domain of mission planning for Computer Generated Forces (CGF), we shall illustrate how we extend the classical planning method by using simulation of more detailed models, where the models simulate entity queuing at fords and bridges as well as engagements. Currently, our engagement simulation uses many of the features of constructive models, such as probabilistic combat results tables, but we will eventually create plans that involve simulated entity-level interaction since this is the most accurate way to learn if a plan will fail or succeed.

3.1 Planner Architecture

Figure 1 displays the architecture of our planner in relation to the Institute for Simulation and Training (IST) Computer Generated Forces (CGF) Testbed [7]. Our mission planner is an integral part of a larger project of the IST called "Intelligent Autonomous Behavior by Semi-Automated Forces in Distributed Interactive Simulation" which is funded by the U.S. Army Simulation Training and Instrumentation Command (STRICOM). The goal of the planner is to automatically derive plans for a semi-automated force or CGF, at the company level initially, so that the force will provide an Army trainee with an effective training experience. Planning is only a small part of the overall project, which includes efficient line of site (LOS) determination, terrain reasoning, intelligent target acquisition and behavior representation for CGF entities. The planner takes orders from the battalion level and translates these orders, with a tight coupling with the terrain analyzer, into efficient plans for the CGF platoon entities. In addition to planning for its subordinate units, the planner must also also be able to monitor the execution of the plan, react to unexpected situations and replan if necessary.

Each commander in the IST testbed is simulated by a Command Entity (CE) whose major functions are performed by the Planner. The planner has two phases: the Reactive phase and the Planning phase. A Phase is a group of states that collectively display a behavior. Only one phase is active at any given time. The starting phase is the Reactive Behavior. Based on the inputs, the current active phase makes the decision as to which phase becomes active next. There is no single 'main' algorithm that controls the whole process. Thus, the decision is made in a distributive manner. We will describe each component of the planner through a demonstration scenario illustrated in figure 2. The friendly company unit 1 (the company entity receiving the Oporder) is situated at Assembly Area(AA) located at (50000, 52500). Company unit 1 is made up of 3 platoons: platoon A is made up of 4 M1 Abrams Main Battle Tanks and each of the remaining 2 platoons B and C are made up of 4 Bradley Infantry Fighting Vehicles. There are two enemy platoons: platoon A is made up of 4 M1 tanks located at (32500, 28750) and platoon B is made up of 4 M2 fighting vehicles located at (45000, 46250). The Operation order given to company unit 1 is to "SEIZE the Objective at (32500, 28750)". The company unit boundaries are given as the rectangular area drawn in the figure. The goal of the command entity is to accomplish the mission with minimal loss of strength.

3.1.1 World Database(DB)

The World Database is not a complete spatial representation of the battlefield (the Terrain Analyzer(TA) has this information) but a simplified database which mainly contains information that is only known to the CE. Since the TA does not have any information regarding the location of enemy or friendly units and does not keep track of the locations, the planner needs to keep track of these locations and the status of the units in the World DB. This database is created as soon as the CE starts to exist. Initially it contains its own location and will be updated with new information as it becomes available to the CE via Sitreps or Oporders.

3.1.2 Reactive Behavior

The Reactive Behavior module displays reactive behavior necessary for survival when immediate action is required. The module is initialized with a generic set of behaviors at the start and may be modified with any reactive behaviors provided by an Oporder.

3.1.3 Planning Behavior

The Planning Behavior module generates orders for its subordinate entities from an Oporder given by a higher level entity. This module is made up of the following smaller modules where the order in which they are presented actually coincides with the algorithm steps of a typical planning process.

1. **Sitrep/Oporder Analyzer** parses the Situation Report(Sitrep) or the Operation Order(Oporder) to update the World DB. In the case of an Oporder, it is further parsed to generate a list of task(s) to be achieved. The Situation Analyzer is called next with this list. In the case of a Sitrep, it is analyzed to decide if any immediate action is required, if any replanning is required, or if any Sitrep needs to be generated. Then, the Execution Monitor is called with the decision.

2. **Situation Analyzer(SA)** uses a set of rules to analyze the given situation using the World DB and performs a set of alternate calls of Route_Request and Define_Tactical_Position producing a number of alternate routes.

3. **Course of Action(COA) Tree Generator**, using the set of alternate routes produced by the SA, generates a COA Tree where the 1st level contains alternative subunit or platoon combinations and 2nd level contains alternative route combinations. The following levels can contain other alternatives such as varying the role of platoons in different formations.

   Figure 3 shows the COA tree generated by the prototype. The SPLIT subtree describes the course of action for a company where the company will be split up. Given that a company has 3 platoons, the number of routes needed is at most 3. If two platoons go one route and one platoon go another, two routes are needed in total. If each platoon goes on a different route, three different routes will be necessary. The 1st level of this subtree will contain all the possible combinations of splitting a 3 platoon company. The heuristic used in this scenario is not to allow Platoon A to travel alone at any time since M1s are considerably slower and lower power than M2s. This restricts the SPLIT combination to 4 sets: (AC,B) (B,AC) (C,AB) (AB,C). From these

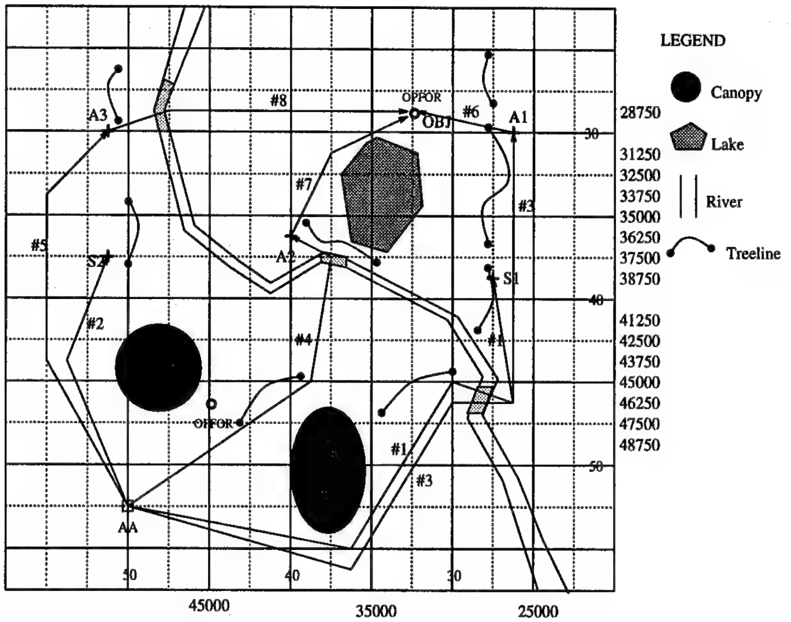


Figure 2: Company mission and routes

combinations, the Create_COA_TREE generates possible combinations of route sets. Since the mission is SEIZE, at least one route should lead the platoons to an ASSAULT_POS. These routes are 3,4,5 according to figure 2. Thus, the possible set of route combinations are (3,1), (3,2), (4,1) (4,2), (5,1), (5,2), (3,4), (3,5), (4,5). The second level of route combinations in the COA tree contains the routes that connect each of the 1st level routes to OBJ if possible. For example, route 6 extends route 3 to OBJ. However, route 1 is not extended to the objective because it's a SUPPORT_BY_FIRE position. The NO_SPLIT subtree has a single subunit combination (ABC) since no split up is allowed in the unit combination. Therefore only a single route set alternatives that lead to an assault positions (3,4,5) are possible. The second level routes are (6,7,8) respectively. No pruning is being done in the current implementation but heuristics can be used for pruning when necessary. Next, the COA Tree Simulator is called with the COA Tree.

4. **The COA Tree Simulator** takes each level of the COA subtree and simulates each route and calculates a score for each friendly platoon per each route. This is done by creating a Simulated World (SIMDB) and performing the simulation of friendly and enemy units by time slicing between actions (move, look, fire) and observation by each unit. In the current version, the enemy unit is simulated in a very limited manner. The enemy unit is assumed to remain stationary and only engage in combat when an opposing force unit has been sighted. The model that is used is the Aggregate Combat Model [2]. An alternative method is to allow the enemy units to have the same planning capabilities as the friendly units but with different tactics. This method would be quite realistic, but it can be quite time consuming. If computing capabilities are limited, we can perform simulation at different levels of abstraction [8, 4] where each higher level will use less computational power. The actual simulation algorithm is as follows:

```
While (planner active) do
    Update entity state variables
    Perform line of sight (LOS) check
    Engagement check
    Update current clock time by $\Delta T$
End While
```

In low mobility areas or areas with a steep terrain gradient, the movement is slower. Also, for some terrain features, as with fords or chokepoints, a simple queuing model can be executed to keep track of entities that must wait for entities that are blocking the path. Service times and speed

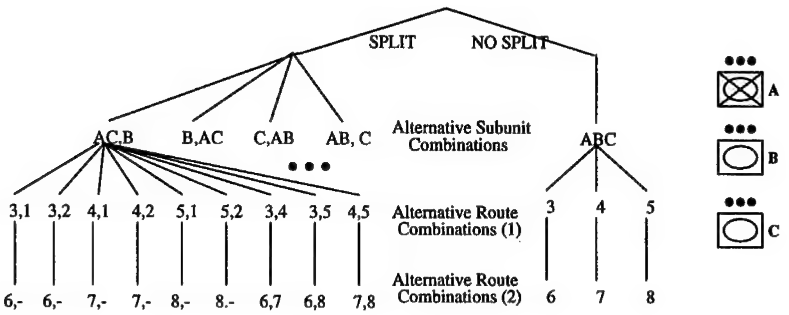

SPLIT
NO SPLIT
AC,B
B,AC
C,AB
AB, C
Alternative Subunit Combinations
ABC
3,1
3,2
4,1
4,2
5,1
5,2
3,4
3,5
4,5
Alternative Route Combinations (1)
3
4
5
6,-
6,-
7,-
7,-
8,-
8,-
6,7
6,8
7,8
Alternative Route Combinations (2)
6
7
8
A
B
C

Figure 3: COA Tree of the Demonstration Mission

values are obtained by sampling from a probability distribution appropriate for the blocked area. A line of sight (LOS) check and range calculation is done between the entity being simulated and known enemy locations. If the enemy is within range of certain weapons (such as a HEAT or Sabot round), an engagement will ensue. The simulation proceeds until either the plan has been fully simulated, or the planner is interrupted.

The simulation result is recorded in the form of an integer number, the evaluation score, which is calculated using the following formula:

$$score = strength\,of\,unit + proximity\,to\,OBJ(\%)$$

The overall simulation strategy is branch and bound. Depending on the order of the calls, however, it is possible to simulate the COA tree in a somewhat depth-first manner.

There are several advantages to using Simulation to predict the results of plan execution.

(a) Simulation provides a uniform method without resorting to adhoc solutions. In simulation, each entity in the environment is simulated in a uniform and consistent manner by using models that represent both the physical and behavioral properties. Thus, simulating a plan is a natural consequence of simulating each of the entities by itself without having to worry about the global state change as a result of each entities action.

(b) Because there is no central reasoning node for the simulation but many individual simulation models for different entities, scalability is a natural consequence. Extendibility is another advantage simulation provides. The effects of adding a new type of entity will be clear, only the behavior models of each entity must be updated to recognize and reason about this new entity.

(c) Similar to how simulation is used for visualization, simulation can be easily used to perform visual playback of how a plan was simulated to explain the planner's decision.

5. The **Execution Monitor**

   The Execution Monitor is the main driver of the Planning Behavior module. It issues a set of chosen subtasks in the plan to each units in an Oporder format and executes its own subtask if there is any. If any Sitrep is received, it updates the world DB with any new information included in Sitrep such as sightings, destroyed units and changed location of units. Then it calls the Sitrep/Oporder Analyzer. If the decision returned calls for immediate action, the control is given to the REACTIVE behavior module. If it calls for replanning, the SA is called to start a planning process with the newly updated World DB. Finally, if the decision is to give up planning at the current level, the CE sends a Sitrep to its higher unit reporting of its current status and waits for further orders.

3.1.4 Expert System

The mini Expert System module contains rules to aid the planning process in making decisions such as choosing routes, choosing best COA tree, performing analysis of situations, Oporders and Sitreps.

4 Interface between Terrain Analyzer and Planner

The Terrain Analyzer is the planner's only source of information where terrain is concerned and thus the planner uses the TA quite extensively during the planning process. The TA is responsible for route planning, finding tactical positions, computing Line of Sight and answering questions about terrain features. The interface between the TA and the planner is established by four types of calls; Route_Request, Define_Tactical_Position, Line_Of_Sight and Terrain_Feature.

5 Demonstration Mission Planning Results

From Figure 2, we observe that any friendly units traveling on route 4 is likely to engage in combat with the enemy unit stationed at (45000, 46250). The friendly unit may not be totally destroyed but considerable amount of strength may be lost during combat and therefore will result in a lower score. Thus, any plan that includes route 4 will have lower scores compared to other plans. The evaluation scores produced by the mission planner for the demonstration mission are listed below in the order of decreasing scores.

For the NO_SPLIT subtree: (Note that routes 1 and 2 are not considered since they end at Support_By_Fire positions.)

```
Route 3 -> 6 : 128.0
Route 5 -> 8 : 128.0
Route 4 -> 7 : 89.8068
```

For the SPLIT subtree with platoon combination AC,B: (Note for route combination 3,1 it means platoons A and C travel on route 3 and platoon B travels on route 1.)

```
Route 3,1 -> 6,- : 158.000000
Route 3,2 -> 6,- : 158.000000
Route 5,1 -> 8,- : 158.000000
Route 5,2 -> 8,- : 158.000000
Route 3,4 -> 6,7 : 152.525711
Route 3,5 -> 6,8 : 144.205093
Route 4,5 -> 7,8 : 116.941719
Route 4,1 -> 7,- : 112.500000
Route 4,2 -> 7,- : 112.500000
```

The planner chooses the plan with the highest score and, in the case above, any one of the four highest score plans can be chosen. A good approach is to choose plans at random in such cases to display unpredictable behavior.

6 Toward an experimental design

Since simulation-based planning is a fairly new concept, we need to perform more in-depth studies through various experiments to analyze the methodology thoroughly. For the experiment, we propose to incorporate simulation-based planning into Soar. The Soar architecture provides a stratified approach to specifying, designing, and building Knowledge-Based systems. The system is first described at the knowledge-level, then at the problem-space level the system is defined in terms of how the task is computationally accomplished, and finally at the implementation level. It supports metalevel reasoning which allows the system to recursively reason about problems. Soar also integrates multiple problem-solving methods and knowledge sources. Soar is well known in the military simulation field through the Soar/IFOR project [10] where Soar is being used to generate the behavior of an automated agent for the Tactical Air Simulation. No extensive mission planning is involved in the project; the goal is to simulate the behavior of a single pilot. A recent attempt by the Soar group to solve the planning problem produced a methodology called Multi-method planning. Realizing that no one fixed method can solve a wide range of problems, they proposed the use of different methods for different problems. We further extend the idea of using different methods by employing simulation-based models within the Soar planning framework. Instead of using rule-based models to simulate and evaluate different candidate plans, we will use models to simulate and evaluate.

We will build two planning systems; one that employs the simulation-based models and one that is entirely rule-based. Then, we will compare the results through several problem domains. First, we will experiment with classical AI problems such as the blocks world problem and the machine shop scheduling problem. We will then experiment with the mission planning problem. The comparison criteria between the two systems are (in no particular order):1) Speed - number of plans generated per unit time. 2) Success - the rate of success of plans. 3) Model complexity - how easy is it to design, build and comprehend the system model? 4) Maintenance - extensibility and modifiability. 5) Reactiveness - ability to react during or after the planning process. 6) Adaptability - ability to adapt planning to dynamic changes of the environment during planning.

In order for the experiment to be valid, we must maintain several variables such as 1) Knowledge: the set of knowledge that is used in both the planners must come from the same source, 2) Data: the data provided to the planner such as Terrain Analysis data must be the same, and 3) Evaluation function: the same objective function must be used to choose the best plan. By successfully maintaining the 3 variables as above, we can ensure the validity of the experiments—allowing us to observe the strong and weak points of the systems.

7 Conclusions and Future Work

Through the design and construction of a C-based simulation module which evaluates candidate plans created by varying the route and subunit combinations, we have shown how we are able to perform planning using simulation. By allowing the use of different simulation models for different domains, the planner has the potential to solve a wide-range of problems. The simulation-based planner runs in real-time on an IBM 486 PC. Our near term plan is to create and run the experiments that was discussed in section 6 to thoroughly evaluate the simulation-based planning method. After these experiments, we plan to apply the simulation-based approach to other areas of planning such as traffic control.

8 Acknowledgment

This work was originally sponsored by the Institute for Simulation and Training under contract #307043.

References

[1] M. Czigler, S. Downes-Martin, and D. Panagos. Fast Futures Contingency Simulation: A "What If" Tool for Exploring Alternative Plans. In *Proceedings of the 1994 SCS Simulation MultiConference*, San Diego, CA, 1994.

[2] Paul Davis. Aggregate Combat Models. Technical report, RAND Corporation.

[3] T.L. Dean and M.P. Wellman. *Planning and Control.* Morgan Kaufmann, 1991.

[4] P. A. Fishwick. An Integrated Approach to System Modelling using a Synthesis of Artificial Intelligence, Software Engineering and Simulation Methodologies. *ACM Transactions on Modeling and Computer Simulation*, 2(4):307 – 330, 1992.

[5] P. A. Fishwick and B.P. Zeigler. A Multimodel Methodology for Qualitative Model Engineering. *ACM Transactions on Modeling and Computer Simulation*, 2(1):52–81, 1992.

[6] P.A. Fishwick. *Simulation Model Design and Execution: Building Digital Worlds.* Prentice-Hall, Inc, 1994.

[7] C.R. Karr, R.W. Franceschini, K.R.S. Perumalla, and M.D. Petty. Integrating Aggregate and Vehicle Level Simulations. In *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, pages 231–239, Orlando, FL., 1993.

[8] J.J. Lee, W.D. Norris, and P.A. Fishwick. An Object-Oriented Multimodel Design for Integrating Simulation and Planning Tasks. *Journal of Systems Engineering*, 3:220–235, 1993.

[9] Soowon Lee. *Multi-Method Planning.* PhD thesis, Department of Computer Science, University of Southern California, 1994.

[10] P.S. Rosenbloom, W.L. Johnson, K.B. Schwamb, and M. Tambe. Intelligent Automated Agents for Tactical Air Simulation: A Progress Report. In *Proceedings of the Fourth Conference on Computer Generated Forces and Behavioral Representation*, pages 69–78, Orlando, FL., 1994.

[11] P.S. Rosenbloom, J.E. Laird, and A. Newell. A preliminary analysis of the Soar architecture as a basis for general intelligence. *Aritificial Intelligence*, 47:289–325, 1991.

[12] E.D. Sacerdoti. *A Structure for Plans and Behaviour.* Elsevier-North Holland, 1977.

[13] A. Tate. Generating Project Networks. In *Fifth Int. Joint Conference on Artificial Intelligence*, pages 888–893, Cambridge, Massachusetts, 1977.

[14] S. Vere. Planning in Time: Windows and Durations for Activities and Goals. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, pages 246–267, 1983.

[15] D.E. Wilkins. Domain Independent Planning: Representation and Plan Generation. *Aritificial Intelligence*, 22:269–301, 1984.

Session 1G:

Planning and Decision Making II

Automated Path Planning for Simulation

Jed Marti and Christophe Bunn*
RAND
1700 Main Street, Santa Monica, California 90407

Abstract

Automated route planning over digital synthetic terrain is of importance to systems involving simulated human entities. Using a route planner whose sole criteria is minimizing distance and elevation change, we analyze the effects of terrain resolution on path quality, and compare human and computer generated paths. We conclude that with some measures, the computer generated route plans are reasonably difficult to distinguish from human generated plans.

1 Introduction

Computer route planning provides the backbone for automating simulated behavior of man and machines on synthetic terrain. This paper addresses the effects of terrain and path resolution on generated path quality and compares the results of automated planning to human generated plans.

We are not so much concerned with the route planning algorithm itself as the data that drives it. Our approach exploits techniques borrowed from Geographic Information System (GIS) technology. We emphasize GIS operations over the limited area covering the intended route. This allows us to tailor routes to meet various requirements. We have experimented with many characteristics such as exploiting available ground cover to hide and avoiding or using terrain features such as roads, rivers, and fords, the experiments covered by this paper exploit only minimal elevation change.

Conventional wisdom is that automated route planning requires very high resolution terrain data. Our experiments begin to quantify the relations between resolution, representation, and human generated plans.

Route planning is an important component of both computer simulations and real world planning. Combat simulations exploit route planning to automate some of the drudge work normally associated with laying down many highly detailed plans or provide automatic reactivity to changed situations. This technology is also applicable to wild fire simulations, disaster response planning, herd and swarm simulation, wilderness road planning and the like [1, 2].

2 Background

The literature provides extensive analysis of automated route planning as a graph manipulation [3, 4, 5, 6, 7, 8]. Our system uses the A* algorithm to compute a route over a uniformly spaced grid using costs between grid points as the prime determinant. Generalizations of the algorithm to polygonal terrain have also been considered [9, 10] and other metrics [11].

Existing GIS based route planning is typically based on network representations such as that in ARC/INFO [12]. Though our task can be converted to a graph representation amenable to processing by such systems, this poses the additional problem of integrating the GIS services into a large simulation system or integrating our model into the GIS.

Similar work has examined robot motion planning [13]. Here the problem relies on much greater resolution and vehicle geometry where space between obstacles becomes a problem amenable to solution by the Voronoi Diagrams [14, 15]. The dual of this problem is the mass movement of troops through terrain with obstacles [16, 17]. Likewise, many modern combat simulations and support tools provide automated route planning to some degree. ODIN [18] provides on-road route planning and limited assistance with off-road planning to minimize river crossings.

Previous work attempted to tune a Neural Network to emulate human paths taken from tracks recorded during actual maneuvers [19]. However, this study was unable to validate routes over terrain that the network did not recognize. Artificial Neural Networks are also used to solve navigation problems by modifying the

*Currently at Logicon Strategic & Information Systems, 222 W. Sixth Street, P.O. Box 471, San Pedro, CA 90733-0471

0-8186-6440-1/94 $04.00 © 1994 IEEE

weights of the different terrain features [20]. A study of reconnaissance planning over polygonal terrain [21] used a measure of success as the time spent to locate targets rather than ensuring that human and machine generated routes looked the same. Our measure of success is making machine generated routes resemble human generated routes.

3 Algorithm Structure

Our route planner requires 3 steps: 1) build a matrix covering route end-points, 2) compute point-to-point movement costs to meet route quality constraints, 3) compute the route plan that minimizes cost. Our examination covers the characteristics of steps 1 and 2. Minimal cost algorithms are covered in the literature.

3.1 The Cost Matrix

We compute the cost matrix by examining terrain data and the path type constraints. We base all paths on minimum distance between two points modified by various path constraints. For the purposes of these experiments we limit the route planner to minimizing elevation changes along the path.

The elevation minimizing hueristic assigns exponentially increasing cost to slope. Tuning this slope constraint to match human performance is the subject of our final experiment. For a slope weight λ, slope s and distance between two points d, we have a cost matrix component $C_{i,j}$ of:

$$C_{i,j} = f(d)e^{\frac{s \ln \lambda}{\beta}} \qquad (1)$$

where $f(d)$ is a linear function of distance, and the constant β is determined to be the greatest slope a wheeled or tracked vehicle can comfortably climb. For the purposes of experimentation, we initially chose $\lambda = 10$, which multiplies the cost by 10 when $s = \beta$.

A typical route plan requires between 10 and 100 intermediate points. We first experimented with an unrotated cost matrix that covers both the start and end points. However, diagonal end points waste space and time by exploring low likelihood routes far from the central route. Likewise, the unrotated matrix restricts the search area near the start and end points for diagonal paths. As seen in Figure 1, rotating the matrix to cover the start and end points results in considerable space and time savings.

To minimize the size of the cost matrix, we determine the best rectangular ratios to cover the path areas. For a potential path between two end points we have the geometry of Figure 2 with the number of extra cells along the new path l, extra cells across the path w and distance d in kilometers between the two end-points P_1 and P_2.

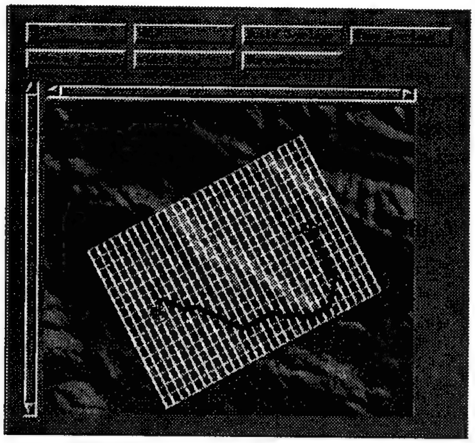


Figure 1: The Rotated Cost Matrix Around Start And Destination

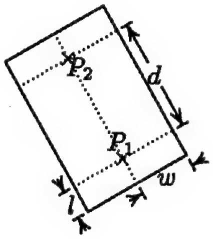


Figure 2: Rectangular Cost Matrix Size

In our experiment, we compute 1000 random paths increasing the size of the cost matrix until the path stabilizes. We establish two important parameters - the extra elements needed along the path axis and extra width needed. Over medium relief topography, for 1000 paths selected at random, we find the maximum required extensions shown in Figure 3. As can be noted, the extra length l is not a function of distance, but w is. We selected,

$$\begin{aligned} l &= 3 \\ w &= \left\lfloor \frac{d}{2r} + .5 \right\rfloor \end{aligned} \qquad (2)$$

to increase the width as a function of distance and the path resolution r. The width must increase because longer paths cover more terrain and have more options for divergence from the straight and narrow. We expect that the extra slop l may need to be increased for rougher terrain than tested.

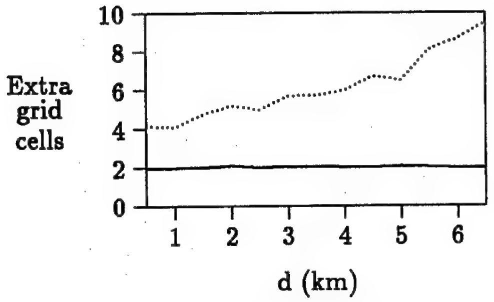


Figure 3: Width vs Height Experiment Over Moderate Relief Terrain

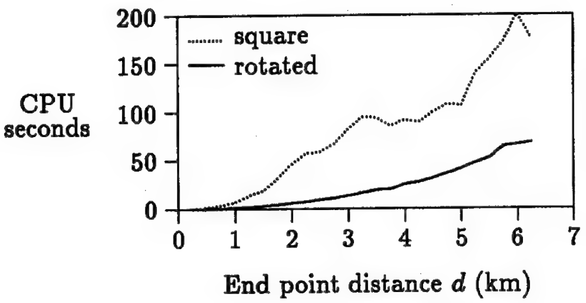


Figure 4: Path Length vs CPU Time

To quantify the advantages of the diagonal approach, we computed 1000 identical path end-points with both rotated and unrotated matrices with the results shown in Figure 4. As can be noted, the rotated version provides superior performance for longer paths beacuse fewer elements of the cost matrix are computed and examined.

4 Path Quality Analysis

Our second set of experiments quantifies path quality by resolution and representation. We attempt to answer the question, does increasing resolution lead to convergence on a particular path and where should the resolution be set? Second, is there a difference between using grid posts (flat tiles) and smoothed (interpolated) data?

4.1 Comparing Interpolated to Grid Post Elevations

We first experimented with two different elevation representations. The simplest form, flat elevation tiles with elevation at any point determined by rounding coordinates to the nearest is speed efficient. The more complex bilinear interpolation computes elevation from the four nearest points. Other research comparing a quadratic spline representation to bilinear interpolation suggests that results are identical but the computation cost of the spline greatly exceeds that of the bilinear interpolation [22].

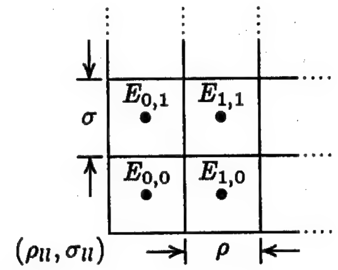


Figure 5: Elevation Grid Tiles

Elevation values are stored in an array $E[x, y]$. Each integer point (x, y) corresponds to an average measurement at a certain latitude and longitude with the dimensions and coordinates shown in Figure 5.

The flat tile computation retrieves the appropriate elevation $e_{X,Y}$ at longitude X, latitude Y by:

$$e_{X,Y} = E\left[\left\lfloor \frac{X-\rho_{ll}}{\rho} \right\rfloor, \left\lfloor \frac{Y-\sigma_{ll}}{\sigma} \right\rfloor\right] \tag{3}$$

with suitable checks to ensure the point lies within the array boundaries.

Bilinear interpolation computes the coordinates of four corner points surrounding the sample point as shown in Figure 6.

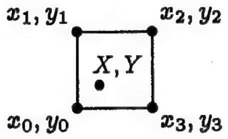


Figure 6: Interpolation Points

We compute elevation $e_{X,Y}$ by the equations:

$$\begin{aligned} x_0 = \left\lfloor \frac{X-\rho_{ll}}{\rho} \right\rfloor, y_0 &= \left\lfloor \frac{Y-\sigma_{ll}}{\sigma} \right\rfloor \\ x_1 = x_0, y_1 &= y_0 + 1 \\ x_2 = x_0 + 1, y_2 &= y_0 + 1 \\ x_3 = x_0 + 1, y_3 &= y_0 \end{aligned} \tag{4}$$

$$\begin{aligned} e_b &= (X - (\rho x_0 + \rho_{ll}))\frac{E[x_3, y_3] - E[x_0, y_0]}{\rho} \\ &\quad + E[x_0, y_0] \\ e_t &= (X - (\rho x_0 + \rho_{ll}))\frac{E[x_2, y_2] - E[x_1, y_1]}{\rho} \end{aligned}$$

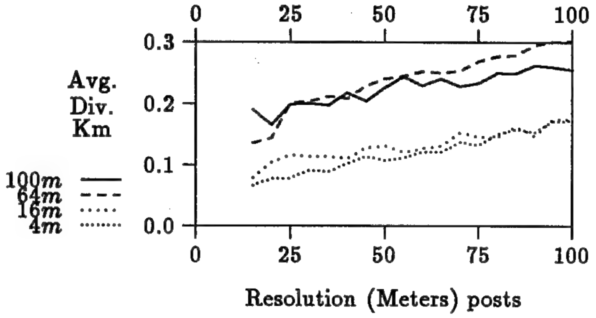


Figure 7: Path Divergence Using Elevation Posts

$$+E[x_1, y_1]$$

$$e_{X,Y} = (Y - (\sigma y_0 + \sigma_{ll}))\frac{e_t - e_b}{\sigma} + e_b \qquad (5)$$

To understand the effects of terrain resolution on these representations we computed many paths and compared their differences.

Our first experiment attempts to determine the effects of terrain and path resolution on computed paths. We vary the path resolution from 10 to 100 meters in steps of 5 for 30 random paths. We also vary the terrain data resolution. For 30 paths selected over a 6 × 6km area of Ft. Hunter-Liggett, we compare path divergence for four terrain resolutions available: resampled 100m DMA DTED data, 64m, 16m and 4m PEGASYS data. We computed average divergence from the 10 meter resolution path to the others for 31 uniformly spaced points on each. That is, if $F(d), G(d)$ are the locations at distance d along the two paths, and $\vec{f}, \vec{g}$ the path lengths, the divergence is:

$$\frac{\sum_{i=0}^{30} \overline{F(\frac{i\vec{f}}{31})G(\frac{i\vec{g}}{31})}}{31} \qquad (6)$$

Using elevation tile data provided the results of Figure 7. As can be seen, the divergence increases with lower path and terrain resolution. Figure 8 shows two example paths with identical end points computed at 25 and 150 meter resolutions over 64m resolution terrain.

However, using the same data with the bilinear interpolating function greatly decreases the effects of both terrain and path resolution. Figure 9 was computed with the same points and paths. Terrain resolution has much less effect on path divergence when using the bilinear interpolating function.

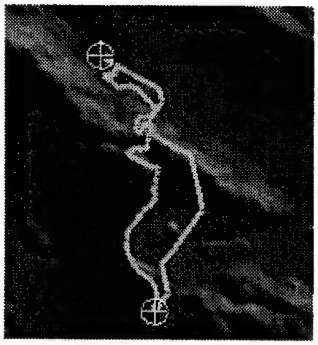


Figure 8: Identical End-Points, 25 and 150 Meter Path Resolution

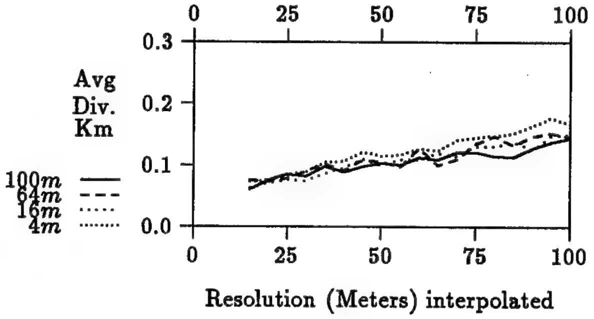


Figure 9: Path Divergence Using Bilinear Interpolated Elevation

4.2 Effects of Relief

We now compare paths generated to minimize elevation changes with both versions to understand if relief has an effect on the path divergence. Over the four subjective terrain types all at 100m resolution we also compute the standard deviation of terrain elevations (the number following the type):

High Relief:242 Mountainous desert terrain (Ft. Irwin Military Reservation)

Medium Relief:35 Semi-mountainous rolling terrain (Ft. Hunter-Liggett)

Medium Relief:39 Rolling terrain with numerous river and creek valleys (Ft. Hood)

Low Relief:38 Flat with small features (Northeastern Saudi Arabia).

We generated 30 random pairs of points and compared the divergence of paths generated with both systems. As before, divergence is the average distance between the highest resolution path and the inspected path for 30 points evenly spaced on both paths. In Figure 10

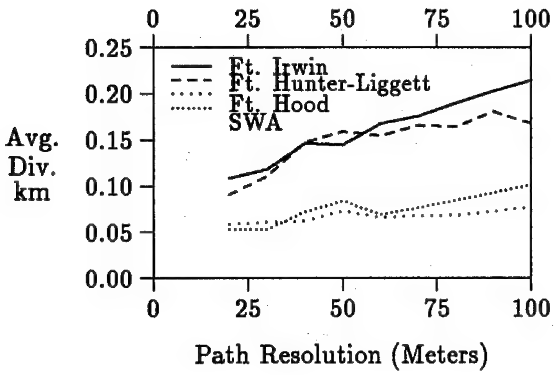


Figure 10: Terrain Relief vs Path Resolution

the divergence is much more pronounced in the rugged terrain of Ft. Irwin and Ft. Hunter-Liggett than the flatter terrains. Unfortunately, the relationship to the roughness measure (the standard deviation of all elevation points) is less clear. We will investigate the Natick terrain classification [23] as a better indicator of terrain roughness.

5 Tuning to Match Human Generated Routes

Comparing our generated routes to actual human performance is extremely difficult. We rejected "in the field" experiments as too expensive and too difficult to control the route parameters. Likewise paths generated during field exercises have hidden assumptions difficult to quantify in our context.

5.1 Experimental Approach

We opted for a two phase approach using two dimensional terrain maps. 23 test subjects were given a short introduction to the task and a number of practice routes to familiarize them with the tool. They were then given 30 start and end points to generate routes and unlimited time. The subjects were never shown either the computer generated routes, a paper map, or told where the selected terrain is. The subjects could backup the route, determine elevations along the route and view a path elevation profile (see Figure 11).

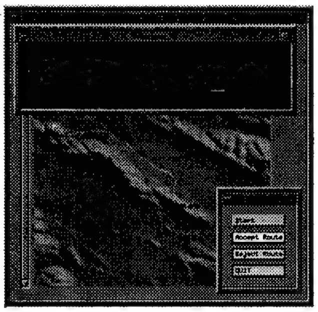


Figure 11: Human Generated Route Planner Display

The first test compared human path planning with elevation and distance the sole criteria. Subjects were told:

> "You are to plan the route between *START* (point at the end of the green line) and *END* (point at the end of the red line) for a medium mobility vehicle for 30 randomly generated sets of points. Your plan should be the most efficient route between the two points (which is not always a straight line). Most efficient in terms of speed, fuel economy, driver fatigue, etc. ...
>
> There are no enemy threats within the area."

We accumulated results from 23 human subjects, 21 males, 2 females, with disparate levels of military and map reading experience. No attempt was made to recruit a random population. 8 subjects had some military map reading and route planning experience. Most of the subjects had some ability to understand contour maps.

5.2 Results

We first must understand the differences among the human planners. If the computer route plans do not differ significantly from humans, we can claim success. We first compare two planners with the most recent military route planning experience with the rest of the subjects. In Figure 12 we compare their 30 routes for average divergence with the other 22 subjects and sort the subjects by average divergence. The top gray bar indicates the maximum divergence range, and the bottom gray bar the minimum divergence range. The second planner had one or more routes with radical differences to all other planners as indicated by a greater maximum divergences for nearly all other planners. This probably indicates that the second planner was planning routes to avoid potential enemy positions. His routes were generally quite different than subjects planning on distance and relief.

We next tune the route planner to minimize its divergence from the 23 human planners. As seen in Figure 13, the least divergence is at $\lambda = 4.0$ (see equation 1).

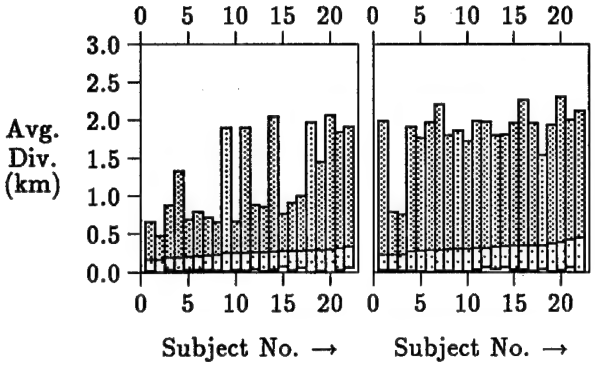


Figure 12: Two Military Planners Compared To Other Subjects

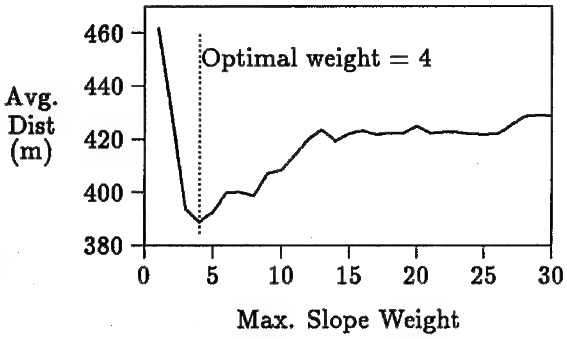


Figure 13: Adjusting Maximum Slope Constant To Match Human Plans

Finally, we compare the average divergence for the $\lambda = 4$ planner with the 23 human routes and a straight line between the end points. Each point is an average divergence between the "player" and all other players including the computer and the straight line. This measure of "averageness" is the degree to which a player differs from all others. As seen in Figure 14, the computer-generated route plans fall nearer the "unique" end of the data, but still better than at least 15% of the subjects and the straight line.

6 Conclusions

From these experiments we draw several conclusions:

1. Minimizing cost matrix size provides a significant reduction in computation time.

2. Using bilinear interpolation of elevation generates routes less sensitive to orientation and data granularity.

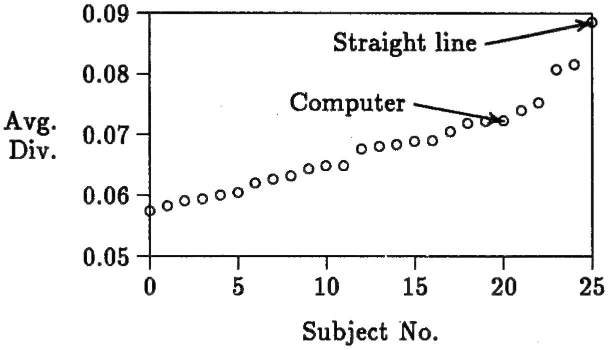


Figure 14: Human vs Computer Route Planning

3. Using higher resolution terrain data provides a linear improvement in path quality for an exponential increase in data space.

4. Increasing path resolution does not increase path quality without a corresponding increase in data resolution.

5. The route planning algorithm must be tuned for different terrain reliefs (the cost matrix must be larger for high relief terrain).

6. The route planner can be tuned to match human performance for some measures.

However, three questions remain to be answered. Can human subjects differentiate between human and machine generated routes (the Turing test)? What measure of terrain roughness can be used to tune the planner's cost matrix size? Will route smoothing minimize differences between human and Machine generated routes [24]?

Our experiments pointed us to a number of incremental improvements and experiments for the planner. First, creation of a "tracked vehicle" elevation cost function that penalizes travel perpendicular to the slope. Second, adding a cost to sharp turns for wheeled and tracked vehicles to minimize the effect of varying path resolution on the generation of switchbacks. Finally, basing the route planner cost matrix size on local contour and data roughness.

References

[1] J. Ross, "An expert system for soil erosion mitigation in logging operations on steep land," *AI Applications in Natural Resources, Agriculture, and Environmental Sciences*, vol. 7, no. 4, pp. 69–70, 1993.

[2] D. S. Mackay, V. B. Robinson, and L. E. Band, "An integrated knowledge-based system for managing spatiotemporal ecological simulations," *AI Applications in Natural Resources, Agriculture, and Environmental Sciences*, vol. 7, no. 1, pp. 29–36, 1993.

[3] E. B. Feinberg, "Characterizing the shortest path of an object among obstacles," *Information Processing Letters*, vol. 31, pp. 257–264, June 1989.

[4] R. Gould, *Graph Theory.* Menlo Park, California: The Benjamin/Cummings Publishing Company, Inc., 1988.

[5] D. B. Johnson, "Efficient algorithms for shortest paths in sparse networks," *J. ACM*, vol. 24, pp. 1–13, January 1977.

[6] N. J. Nilsson, *Principles of Artificial Intelligence*, ch. 7. Palo Alto: Tioga Publishing Co., 1980.

[7] S. Ntafos, "The robber route problem," *Information Processing Letters*, vol. 34, pp. 59–63, March 1990.

[8] R. A. Wagner, "A shortest path algorithm for edge-sparse graphs," *J. ACM*, vol. 23, pp. 50–57, January 1977.

[9] C. T. Cunningham, "Control of movement in an arbitrary polygonal terrain," in *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, pp. 307–315, Institute for Simulation and Training, March 1993.

[10] J. S. Mitchell, "An algorithmic approach to some problems in terrain navigation," *Artificial Intelligence*, vol. 37, pp. 171–197, December 1988.

[11] K. Kanchanasut, "A shortest-path algorithm for manhattan graphs," *Information Processing Letters*, vol. 49, pp. 21–25, January 1994.

[12] Environmental Systems Research Institute, Inc., 380 New York Street, Redlands, California, *Understanding GIS The ARC/INFO Method.*

[13] C. K. Yap, "Algorithmic motion planning," in *Advances in Robotics: Volume 1* (J. T. Schwartz and C. K. Yap, eds.), Hillsdale, NJ: Lawrence Erlbaum Associates, 1987.

[14] G. Voronoi, "Nouvelle application des parameters continus à la théorie des formes quadratiques. Deuxieme mémoire, recherches sur les paralleloedres primitifs," *Z Reine Agnew Math*, vol. 134, pp. 198–287, 1908.

[15] M. S. Chang, N.-F. Huang, and C.-Y. Tang, "An optimal algorithm for constructing oriented voronoi diagrams and geographic neighborhood graphs," *Information Processing Letters*, vol. 35, pp. 255–260, August 1990.

[16] L. R. S. Alexander, "Intelligent application of artificial intelligence," *PHALANX*, pp. 20–23, December 1991.

[17] J. B. Marti, "Cooperative autonomous behavior over large scale terrain," in *AI, Simulation and Planning in High Autonomy Systems*, reprinted as RAND N-3130-DARPA, IEEE, March 1990.

[18] T. Stanzione, J. E. Smith, D. L. Brock, J. M. F. Mar, and R. B. Calder, "Terrain reasoning in the ODIN semi-automated forces system," in *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, pp. 317–326, Institute for Simulation and Training, March 1993.

[19] T. Bui, D. Dryer, and M. Laskowski, "A neural-network based behavioral theory of tank commanders," Tech. Rep. NPS-AS-92-015, Naval Post Graduate School, May 1992.

[20] M. A. Penna and J. Wu, "Models for map building and navigation," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, pp. 1276–1301, September/October 1993.

[21] D. R. V. Brackle, M. D. Petty, C. D. Gouge, and R. D. Hull, "Terrain reasoning for reconnaissance planning in polygonal terrain," in *Proceedings of the Third Conference on Computer Generated Forces and Behavioral Representation*, pp. 285–305, Institute for Simulation and Training, March 1993.

[22] D. A. Marlin, "Digital terrain evaluation study," tech. rep., Hughes Aircraft Company, P.O. Box 80028, Los Angeles, California, 1992.

[23] US Army Material Systems Analysis Activity, "The Natick landform classification system," Tech. Rep. 100, US Army, Aberdeen Proving Ground, MD, October 1974.

[24] M. Shah, K. Rangarajan, and P.-S. Tsai, "Motion trajectories," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 23, pp. 1138–1149, July/August 1993.

A Distributed Simulation System for Team Decisionmaking

Alan A. Song and David L. Kleinman
Department of Electrical and Systems Engineering
The University of Connecticut
Storrs, CT 06269-3157

Abstract

This paper gives an overview of a unique distributed, real-time simulation system for studying team decisionmaking and coordination - the DDD (Distributed, Dynamic, Decisionmaking) paradigm. The DDD paradigm captures the essential elements in real-world decisionmaking problems and integrates them into a controlled, computer-mediated, laboratory setting. The DDD simulation system is implemented on a network of UNIX workstations with real-time control, on-line data acquisition, interactive graphical display, and a simulated inter-human communication network. With a highly reconfigurable user interface and a flexible scenario generator, DDD has been used in many team decisionmaking experiments with different problem context, including military command and control, job scheduling, and medical diagnosis.

1 Introduction

In large scale systems that involve humans, machines, computers, etc., the problem scope and complexity often requires that the decisionmaking function be distributed over several humans. Quite often such systems have a team of human decisionmakers who are geographically separated, but who must coordinate to share their information, resources and activities in order to attain common goals in what is generally a dynamic and uncertain task environment. Although problem contexts can be different among various systems (e.g., military command and control, electric power distribution, air traffic control), the essential elements of decisionmaking remain the same. In order to study problems such as those above on a scientific basis, we have developed a unique distributed simulation system, termed DDD (Distributed Dynamic Decisionmaking) paradigm, that abstracts and simulates the essential elements of real world decisionmaking problems.

Unlike existing large scale simulation systems such as SIMNET [18] which stresses high fidelity, specialized, full scale simulation, the DDD paradigm stresses the small team (typically with less than ten team members) with an abstracted, low fidelity task environment, and emphasizes the basic aspects of interaction and coordination that are central to "teamness". It simulates the real-world problems in such a manner as to be amenable to study in a controlled laboratory setting. The task environment in DDD is reconfigurable for different problem context. For example, in our previous research, the system has been configured as naval command and control, military situation assessment, medical diagnosis, job scheduling in manufacturing systems, etc. The DDD system can be used as a versatile tool for studying/training small teams in military or industry.

The DDD paradigm is built upon the body of knowledge we have accumulated during the last ten years in performing model-driven, basic experimental research [1] [2] [3]. As the backbone of our normative-descriptive research for team decisionmaking and coordination, the DDD paradigm has been used for more than fifteen team-in-the-loop experiments, and proved to be a very powerful empirical research and training tool [6]-[17]. The DDD paradigm is implemented on a network of UNIX workstations, with facilities providing real-time control and on-line data acquisition, an interactive display/interface media, and a computerized inter-human communications and information network within which delay and occasional failure can be manipulated. The simulation system can run on workstations connected by a local area network, or on remote workstations connected by Internet. Its X11/Motif based graphical interface is highly reconfigurable (viz., for different problem context, the look and feel can be very different). Currently, it can support up to seven-person hierarchical or parallel team (expandable if desired). The DDD simulation system has the flexibility to examine a variety of ways in which information processing and resource allocation prob-

0-8186-6440-1/94 $04.00 © 1994 IEEE

lems can be solved by a team of decisionmakers (DMs) under different organizational architectures and information structures.

This paper gives an overview of the DDD simulation system with an emphasis on newly developed features that are not included in our early report [3]. The remainder of this paper is organized into two sections and a conclusion. In section 2, the team decisionmaking environment is described, and the basic elements of the DDD paradigm including resources, tasks, information, and responsibility are discussed. Section 3 reviews the main features of the simulation system including system architecture, user interface, scenario generator, experimental variables, built-in distributed database, and training support tools. Finally, section 4 offers concluding remarks.

2 Basic Elements of DDD paradigm

The DDD paradigm is implemented as a computer-driven interactive game among several decisionmakers (DMs) who may be geographically separated (see Figure 1). In a real time simulation session, each DM sits at a workstation which is capable of displaying the tactical situation and sending/receiving information to/from the other players. Team decisionmaking is formulated as a process of allocating limited resources to a variety of tasks in a dynamic and uncertain environment. Thus, the essential elements of team decisionmaking are abstracted as: i) resources (e.g., machine tools, man powers, sensors, weapons, etc.), ii) tasks (jobs to process, e.g., parts, unidentified targets, enemy airplanes, etc.), iii) information (e.g., sensor measures, intelligent sources, reports, etc.), and iv) responsibility (i.e., who should do what, at what time). To achieve the team goal, co-acting DMs must process distributed information to: i) estimate/identify various task attributes, and ii) determine and schedule their resources to process specific tasks. The DMs are thus required to coordinate their information, actions, and resources in a timely and accurate manner. Below we describe in more detail the salient elements of the DDD paradigm.

2.1 Resources

Resources are basic elements of the system. A resource can carry other resources called sub-resource, for example, a destroyer can carry some helicopters, and the helicopters can carry some sonobuoys, etc. In this way the resources can be nested down to any desired level of detail.

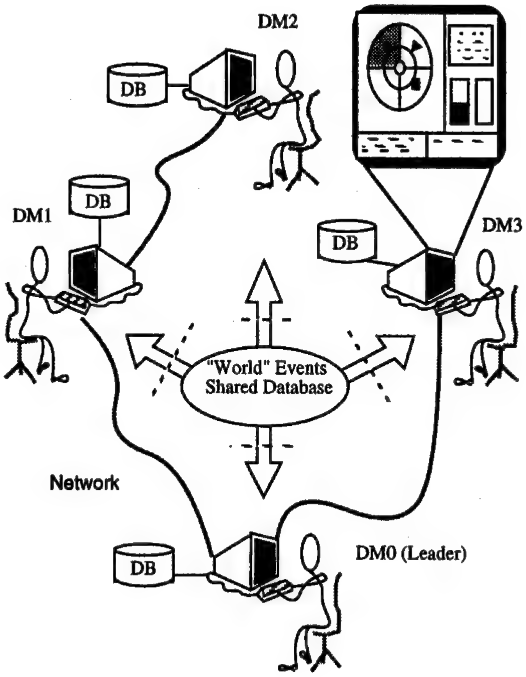


Figure 1: The Distributed Dynamic Decisionmaking Environment

The resources are divided into several classes depending on the design parameters of the experiment. All resources of a given class will have the same features with respect to capacities (i.e., sensor range, weapon strength, etc.). The only difference among resources in a given class is the number of sub-resources each carries.

The sub-resources are located on board their parent resource. A sub-resource does not become an independent resource until it is launched from the parent resource. The DM can launch one or more sub-resources that will become available after a certain launch time delay. The sub-resources can only stay away from their parent for a limited time period. An industrial example of resource/sub-resource could be the manager (resource) that hires temporary employees (sub-resources).

The strength of a resource can be described as a generalized vector which stands for strength in different aspects. Each resource has its effective range. For example, a sensor resource can have three ranges: a detection zone, a measurement zone, and a classification zone.

Each resource is controlled by the DMs who own it(a resource can be owned by multiple DMs), and the control of any resource may be transferred during the simulation from one DM to another with an attendant transfer time delay.

2.2 Tasks

A team is presented with multiple tasks having different deadlines, processing times, attributes, and priorities. During the real time experiment, tasks appear, move/maneuver and disappear according to a scenario that is under the control of the experiment designer.

The tasks are also divided into classes. For example, we can have AA, AB, AN which may correspond to different air targets such as a backfire bomber, a bird, or a civilian airplane. The hostility of each task class, i.e., whether they are threats or neutrals, can be defined by the experiment designer.

Each task has an attribute vector a with elements that characterize it quantitatively. For example, the attributes can include strength, evasiveness, vulnerability, etc. These attributes are random from task to task, but have a probability distribution (mean and standard deviation) that is unique to task class. The resources r required to successfully process a task is a mapping of the attributes of that task and will generally depend on task class.

Tasks can be processed in one or more operations, each operation can be assigned to different DMs. Two types of processing are possible: sequential and parallel. The sequential processing requires two or more DMs to process in sequence, the next operation cannot be started before the current one is finished, for example, a part in a manufacturing line may need molding, painting, and assembling. The parallel processing requires two or more DMs (or resources) to process at the same time, all required operations must be synchronized to complete the processing, for example, to diagnose a disease, all blood test, X-ray, and urine test must be finished before the final decision can be made.

The DDD also includes complex tasks such as active tasks and dynamically attributed tasks. An active task can change its trajectory according to the current situation and the treatment it received. A dynamically attributed task can change its attributes as a function of time and/or location of the task (for a simple task, the trajectory and true attributes are set by the scenario generator and remain unchanged during the real-time session). These complex tasks provide facilities to investigate team decisionmaking and coordination issues in more complex and reactive task environment.

2.3 Responsibility Structure

The overlap in task processing responsibilities of the team members can be adjusted based on the experimental condition. Responsibility can be preassigned in a variety of ways, e.g., by task class or by geographical location. Under the conditions of no overlap we have a disjoint team requiring no coordination in task processing. As overlap is increased, conflicts in the overlapping areas of joint responsibility will occur which will need to be resolved through coordination. A new feature of the DDD is the ability to modify online task responsibility on a task-by-task basis. Thus, the responsibility for individual task prosecution can be (re)assigned dynamically by the team leader.

2.4 Information and Communication

Information and communication are two major aspects in team decisionmaking and coordination. Different structures of information/communication and their impacts on decisionmaking are important research issues. The DDD paradigm provides a variety of mechanisms to manipulate information and communication.

The information structure of the team can be manipulated easily via the DDD paradigm. This is implemented by establishing an information network within the simulation system. Every DMs can be assigned a level of “tie-in” to the information network depending on different task. A high level of “tie-in” means that the DM can get almost all measurements obtained by other DMs, and a low level of “tie-in” means that the DM can only rely on his own resources. Therefore, a centralized, a partially centralized or a decentralized information structure can be accomplished by setting different network “tie-in” levels by task type for different DMs. Furthermore, different roles in a hierarchical team may have different levels of information aggregation. For example, a team leader may have information on overall situation without details, in contrast, a subordinate may have information on detailed local situation within his responsibility.

Communication among DMs is the major way in which the team members can share their local information, and coordinate actions on resource transfer and task processing. In DDD paradigm, communication between different DMs is mainly carried out by electronic messages (Verbal exchanges based on multiperson communication/recording system can also be incorporated to the DDD paradigm, see [16]). In order to simplify the data analysis, all electronic message are preformatted. Our underlying model for the communication channel contains a variety of features that are important to human decisionmaking. To simulate the communication and data processing delay in real situations, a (random and/or fixed) time delay in message transfer was introduced. To simulate the limitation on communication capacity (or channel access), the number of communications (N) in a fixed time window (T) can be specified. Message loss and information scramble due to the network failure can also be simulated within the DDD. Finally, the communication network structure can be defined by a communication matrix, i.e., who can communicate with whom.

3 The Features of DDD Paradigm

3.1 System Architecture

The general architecture of the DDD environment is shown in Figure 2. The DDD paradigm runs on a network of UNIX workstations. In a real-time simulation session, eight (or more) processes run concurrently on different workstations, with all of the control and communication information traffic carried over network. In the figure, Global is a process that works as the "control center" for the environment by controlling the clock and timing, synchronizing the other processes, and sending out various control messages according to the experimental scenario. Each Local Process controls execution within a workstation (WS) and interacts with the User Interface and the Global Process. Each User Interface receives commands from a DM and displays the dynamic tactical situation. The Scenario Generator is used for assisting the experimental designer in developing various system parameters for a given experiment.

In the DDD environment, the global and local processes are implemented via a message-passing approach. Each action of the DM is composed of certain events transferred in the form of messages. For example, when a display object receives a "process" command issued by DM through a mouse/keyboard event, it sends a message "PROCESS EVENT" to a local database object that triggers the method "process" which in turn sends a message to the global process and then other local processes to update the state of all relevant objects. Thus, synchronization is achieved via the LIFO queueing and processing of messages.

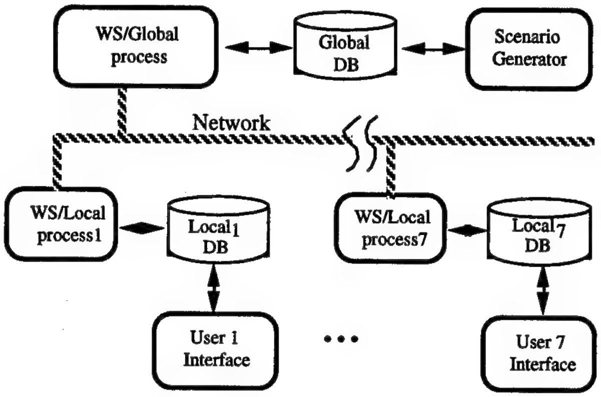
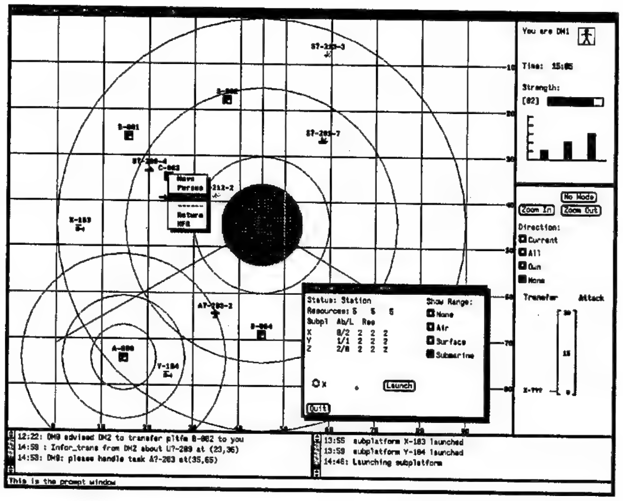


Figure 2: The Architecture of the DDD Paradigm

3.2 Interactive Display

The user interface in DDD is very flexible. While all the facilities for decisionmaking are basically the same, the look-and-feel can be different according to different problem contexts of the scenario. Some examples of display at an individual node are shown in figure 3 and figure 4. Figure 3 is the screen of our basic paradigm [2] [3], three types of targets are shown on the screen: air, surface, and submarine; the resources are ships and airplanes, and sub-resources are helicopters. Figure 4 is a screen form our REST (Reward Structure) experiment [11], where triangles and circles represent targets assigned to different decisionmakers (combined triangle and circle means two DMs are responsible for the target). In these figures, the screen is divided into four major parts: the main display, the status panel, the communication panel and the prompt panel. The main display displays the objects that represent resources and targets. Different targets arrive and move according to a experimenter-defined scenario; targets must be processed within a limited time window. All commands related to the objects can be issued by pull-down menus, pop-up windows or double- click associated with the objects. The communication panel is composed of an incoming window which is used to display the messages from other players, and an outgoing window which is used to display the feedback information when a message is sent out. The status panel is used to display the current time, strength and the dynamics of resource transfer/utilization. The prompt panel is used to display prompts or error messages. All the display icons, menus, windows and messages can be modified or tailored according to different experimental designs.



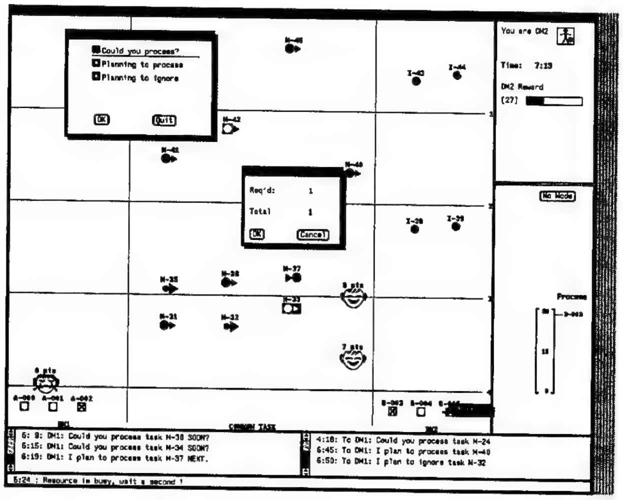
Status: Station
Resources: 5 5 5
Subpl Ab/L Res
X 0/2 2 2 2
Y 1/1 2 2 2
Z 2/0 2 2 2

```
12:22: DM0 advised DM2 to transfer pltfm B-002 to you
14:58 : Infor_trans from DM2 about U?-289 at (23,36)
14:53: DM0: please handle task A?-283 at(35,65)
```

```
13:55  subplatform X-183 launched
13:59  subplatform Y-184 launched
14:46: Launching subplatform
```

This is the prompt window

Figure 3: The Screen of the Basic DDD paradigm



```
6: 9: DM1: Could you process task N-30 SOON?
6:15: DM1: Could you process task N-34 SOON?
6:19: DM1: I plan to process task N-37 NEXT.
```

```
4:18: To DM1: Could you process task N-24
6:45: To DM1: I plan to process task N-40
6:50: To DM1: I plan to ignore task N-32
```

5:24 : Resource is busy, wait a second !

Figure 4: The Screen of the REST Experiment

3.3 Scenario Generator

A scenario generator has been developed to assist the user in setting up the experiment. It is used to configure the resources, to define the tasks, and to design the movements of tasks. The scenario generator is capable of representing a stochastic and imperfectly known environment. For example, unexpected or low probability events can be introduced, and false information and/or false threats can be employed to perturb the system. The intention is to represent a world that is difficult to predict, in which a hostile adversary introduces uncertainties into one's estimation of the current state of the system, thus making inferences about future states rather unpredictable. All task arrival times, task arrival positions, and task movements can be either automatically generated according to a certain random function, or a certain pattern, or specifically designed on a task by task basis.

Two interfaces are provided for the scenario generator. The first one is a flexible experiment description language, XS language, which can be used to define the "rules" of the DDD game, describe the resource and the task environment. Three types of items can be described via XS language, they are: 1) general items; 2) resource information; 3) task information. In general items one can set the overall features of the experiment, such as the numbers of DMs, simulation time and communication delay etc. In resource information one can describe the characteristics of the resources such as maximal velocity, strength, and ranges, etc. In task information one can define task attributes, the resource required to prosecute the task, the decision-makers who are able to see or process the task, etc.; one can also describe the task arrival times, initial positions, velocities, and the maneuvers of the tasks.

A graphical active database modeling tool for scenario generation has also been developed [5]. This tool has utilized data modeling techniques to correctly and precisely specify large amount diverse, intricate, and interdependent information including the structure of the decision team, the sharing of data, the interaction and exchange of data among DMs, and the data required by the different DMs. Furthermore, the structural information can be graphically specified by the experimenter, and changes in structural information automatically cascades to investigate changes of related information throughout the experimental scenario, resulting in time saving and consistent design.

3.4 Experimental Variables

The DDD paradigm is powerful enough to manipulate a variety of independent variables(IVs) that allow for the study and evaluation of different command and control configurations. Some of the major IVs are: i) internal variables (team structure, responsibility structure, information structure, and communication structure), and ii) external variables (tempo, uncertainty, resource quantity, information quality).

The number and type of dependent variables (DVs) this paradigm can handle is quite flexible. To date, over 100 performance, strategy, coordination, and workload measures have been collected and analyzed in various experiments.

All essential operations taken by the DMs are recorded in a log file. This file can be used to generate various dependant variables and statistics. Another important function of this file is that it can be used in play back mode to automatically replay the game for review.

3.5 Distributed Database

The DDD paradigm includes different resources, tasks, coordination tools, decision tools, display tools, and an on-line data acquisition tool. All of these aspects involve different kinds of data that are too complex to manage without using proper database techniques. The requirements for the distributed database can be illustrated by the following DDD features:

- *Response Time Requirement:* Dynamic decisionmaking actions usually have stringent response time requirements. The conflict between time limit and the system's response becomes more significant due to the network communication delays and the time necessary to update many graphical displays and data items.

- *Concurrency Control:* Several DMs may frequently read/write some related database resources concurrently. Thus, concurrency control under time pressure is critical to the system.

- *Low Computational Overhead:* The DDD environment is both compute-intensive and data- intensive. The main body of the environment is devoted to simulation. Thus, the computational overhead of a database management mechanism must be kept low so that the system can properly handle the real-time decisionmaking actions.

- *Complex Data Types:* The DDD environment requires a database that handles complex data types, captures the structure of the data, and considers the operational semantics of the data objects.

The built-in database in DDD has considered all above aspects. To meet the real-time requirements, the database in DDD was designed as a partially replicated distributed database with a priority based transaction management mechanism incorporated. To handle the shared access requirements, we used a hybrid method that combines a priority based concurrency control policy with a certain distributed locking mechanism, so that the system can process transactions within soft deadlines while guaranteeing that the data consistency is not violated. To handle complex data types, we used the object-oriented data model which has a clear advantage over the classical data models, particularly from the perspective of conceptualizing the information and transitioning from the conceptualization to an implementation. The database management mechanisms we have used has proved to be very effective in supporting our real-time distributed dynamic decisionmaking experiments [4] [5].

3.6 Training Support Features

A global control panel was developed so that the experimenter can control the pace of an experiment. Using the control panel, the experimenter can pause or continue the scenario, speed up or slow down the game clock. A play back mode with fast play and slow motion capability was also built in. These features provide useful tools for instructors. According to different training requirement, the feedback information can be designed based on the dependent variables that collected and computed on-line. The information can be chosen to be displayed on a task by task basis, on a per game basis, or by time period, with graphical and/or numerical form. Because of the large amount of dependent variables collected, we can choose the optimal feedback information among them according to different instructional needs. These features have been proved to be very effective for subject training in our past experiments.

4 Conclusion

The DDD paradigm we have developed is a generic paradigm that characterizes team decision processes in which limited, shareable resources must be allocated to identify and process tasks in a dynamic and uncertain environment. It is a research/training tool amenable to systematic and scientific study while retaining the essential features of real-world tactical decisionmaking. The DDD paradigm has been used extensively in our research. In over fifteen experiments, DDD has been proved to be a flexible, easy to use, yet versatile tool for studying distributed decisionmaking in small team configurations. It has enabled us to empirically study numerous issues and test model-driven hypotheses in distributed decisionmaking and coordination. The DDD paradigm is currently used as the main test-bed for our research that studies the adaptation of organizational structure to task environment which is an important step in reaching longer term goals such as the development of a computational theory of organization design, and the understanding of how to design organizations of intelligent agents for high performance.

Acknowledgements

The work reported here was supported in part by the Office of Naval Research under contracts N00014-90-J-1753 and N00014-93-I-0793.

References

[1] D. L. Kleinman, D. Serfaty, and P. B. Luh, "A Research Paradigm for Multi-Human Decision Making," *Proc. 1984 American Control Conference* , pp. 6-11.

[2] A. Song, D. L. Kleinman and D. Serfaty, "A Research Paradigm for Studying Naval Team Decisionmaking," *Proc. 7th Annual Workshop on Command and Control Decision Aiding*, April 1990.

[3] D. L. Kleinman and A. Song, "A Research Paradigm for Studying Team Decisionmaking and Coordination," *Proc. 1990 Symposium on Command and Control Research*, June 1990, pp. 129-135.

[4] A. Song, S. A. Demurjian and D. L. Kleinman, "Transaction Management and Object-Oriented Modeling in a Distributed Dynamic Deisionmaking Environment," *Proc. 1994 ACM Computer Science Conference*, March 1994.

[5] S. Demurjian, M.-Y. Hu, D. L. Kleinman and A. Song, "ADAM/DDD - An Application- Specific Database Design Tool for Dynamic Distributed Decisionmaking," *Proc. IEEE International Conference on Systems, Man and Cybernetics*, October 1991, pp. 2079-2084.

[6] J. Shi, P. B. Luh and D. L. Kleinman, "A Normative-Descriptive Study of Information and Command Strategy in Distributed Team Resource Allocation, Part 1: Experiment Design," *Proc. 1990 Symposium on Command and Control Research*, June 1990, pp. 48-53.

[7] W. P. Wang and P. B. Luh, "Task Sequencing within a Distributed Human Team: Experimental Paradigm and Normative Modeling," *Proc. IEEE International Conference on Systems, Man and Cybernetics*, Nov. 1990, pp. 413-417.

[8] P. Nodoushani, D. L. Kleinman and D. Serfaty, "Performance Feedback and Cooperation in Teams," *Proc. 1991 Symposium on Command and Control Research*, June 1991, pp. 140-148.

[9] J. A. Effken and R. E. Shaw, "Coordination in an Intensive Care Setting," *Proc. 6th International Conference on Event Perception and Action*, August 1991, pp. 289-292.

[10] A. Pete, C. Rossano and K. R. Pattipati, "Distributed Binary Detection with Different Local Hypotheses," *Proc. IEEE International Conference on Systems, Man and Cybernetics*, Oct. 1991, pp. 2023-2028.

[11] P. Shi, P. B. Luh and D. L. Kleinman, "Modelling Human Distributed Decisionmaking with Individual Objectives," *Proc. 30th IEEE Conference on Decision and Control*, Brighton, UK, Dec. 1991, pp. 1225-26.

[12] T.-M. Tang, P. B. Luh and D. L. Kleinman, "Coordination with Constrained Resources: A Modeling Framework," *Proc. American Control Conference*, June 1992, pp. 1978-1979.

[13] J. N. Lin, A. Song, D. L. Kleinman and P. B. Luh, "Hierarchical Team Coordination Under Task Uncertainty: Experiment Design and Petri Net Modeling," *Proc. 1992 Symposium on Command and Control Research*, June 1992, pp. 57-64.

[14] V. Raghavan, K. R. Pattipati and D. L. Kleinman, "Partial Observability and Information Coordination in Teams (POINT): Experiment and Modeling Framework," *Proc. 1992 Symposium on Command and Control Research*, June 1992, pp. 115-120.

[15] K. A. Majalian, D. L. Kleinman and D. Serfaty, "The Effects of Team Size on Team Coordination," *Proc. IEEE International Conference on Systems, Man and Cybernetics*, Oct. 1992, pp. 880-886.

[16] S. P. Kalisetty, D. L. Kleinman, D. Serfaty and E. E. Entin, "CHIPS: Coordination in Hierarchical Information Processing Structures - Experiment and Modeling Framework," *Proc. 1993 Symposium on Command and Control Research*, July 1993.

[17] A. Pete, K. R. Pattipati and D. L. Kleinman, "Team Relative Operating Characteristic: A Normative-Descriptive Model of Team Decisionmaking," *IEEE Trans. on Systems, Man and Cybernetics*, Vol. 23, No. 6, Nov./Dec. 1993, pp. 1626-1648.

[18] C. M. Kanarick, "A technical Overview and History of the SIMNET Project," *Proc. SCS Multiconference*, Jan. 1990, pp.104-111.

The Fire Support Automated Test System (FSATS): An Approach to Distributed Command and Control Simulation

Martin D. Howard
Applied Research Laboratories
The University of Texas at Austin

Abstract

The Fire Support Automated Test System (FSATS) is being developed by the Program Manager, Instrumentation, Targets, and Threat Simulators (PM ITTS) to provide a full suite of instrumentation capabilities to support the technical and operational testing of United States Army Fire Support Command and Control (C^2) Systems. The keystone of FSATS is the capability to simulate the tactical messaging of selected C^2 nodes and to distribute this simulation over geographically dispersed hardware platforms to obtain a realistic distribution of tactical processes and communications. This simulation is based on a series of behavioral models specifically developed for use within this system. The FSATS simulation analyzes the content of tactical messages received and generates an appropriate tactical message in response. While principally developed for the test support role, the benefits and ease of extending this simulation technology into the training role is clear.

1. Background

Recent years have seen an explosive growth in the use of simulation technology to achieve low cost solutions to training problems within the Department of Defense. Attempts to extend this technology into other technical areas has met with only moderate success, since a significant amount of the emphasis in training simulation has, in the past, focused on the man-machine interface necessary to conduct individual training. This feature, although essential in the individual training role, assumes a lower priority when applying simulations to service applications such as collective training, system testing, and evaluations.

Once such application utilizes simulation as an effective cost reduction mechanism in support of the technical and operational testing of today's evolving U.S. Army Command and Control (C^2) systems. The Army's development of automated C^2 systems is critical to meet the increased tactical efficiency demanded of today's force in the face of the changing threat and continued downsizing. These C^2 systems are tactically employed in a network of cells (or nodes) across the operational area, and each is intended to perform a specific function or series of functions to support the overall battle. Using tactical intelligence and information to provide the basis for decision-making, C^2 nodes are data-driven and utilize tactical messaging as the principle means of exchanging information.

Since tactical communications provide the primary data path between nodes, the simulation of any portion of the C^2 network must support this data exchange. Hence, a simulation system which emphasizes a visual-based simulation does not apply. For these reasons, the development of FSATS pursued a simulation based on the tactical messages used by the target C^2 system. The concept of message-based simulation is certainly not founded with FSATS, but this was the first application of this simulation approach to support a distributed and reusable simulation.

Past simulations developed to support system testing and training were hosted on a single mainframe system which contained multiple established communications pathways to the tactical C^2 system, normally referred to as the "System Under Test" or SUT. Simulation of this type were rigid and confined, normally supporting a single exercise configuration in a controlled environment. While suitable for technical testing with predefined goals, this type of simulation does not readily support the needs of

ISBN 0-8186-6440-1. Copyright (C) 1994 IEEE. All rights reserved

0-8186-6440-1/94 $04.00 © 1994 IEEE

the operational tester or the trainer, who relies upon tactical realism and multiple excursions and configurations to fully assess the suitability of the overall C^2 system.

FSATS was designed to provide a distributed, message-based simulation which could be employed in the exercise environment in a manner similar to that used for the target C^2 system. Therefore, the FSATS design incorporated features to address these critical functional needs and to provide reusability and reconfigurability to it user.

2. Message-Based Simulation

The simulation domain of the FSATS system will eventually encompass all tactical messages utilized by existing and planned Fire Support C^2 nodes. A careful analysis of the functional requirements of each node within this message domain will result in a finite set of messages with which the simulation for each node will need to interact. These messages may be categorized as either input messages, to which the simulated node must respond, or as output messages, which form the individual portions of the simulated node's response.

A key design goal of tactical C^2 systems is to automate the routine extraction of data from incoming messages and to manipulate that data into a form which may be clearly presented to the human operator of the node as a decision-making aid. These manipulations may take several forms, such as rearranging or expanding of text for display, conversion of text into graphical displays, or performing routine functions designed to replace repetitive processes normally performed by the operator. Therefore, any simulation of a C^2 node's behavior must properly account for all aspects of this automated data handling, as well as suitably represent the human decisions and actions which would normally occur. In addition, the output message(s) normally generated as a result of this decision must be correctly populated with data, converted to an appropriate syntax, and properly presented to the message handling protocol.

Groups of tactical messages between nodes relate to a single mission being performed in support of the overall battle. This group of messages represent a single Fire Support mission, commonly referred to as a "mission thread". Each of these tactical messages consists of several pieces of tactical information or data which define the ongoing state of missions flowing through the overall system. The nodal simulations must trap and record this information to be able to access the current state of each individual mission. Thus, each nodal simulation makes use of a series of tactical state tables to record the dynamic data associated with each mission. This same data is also used to populate the output messages generated by each model.

3. The Nodal Logic Model

The FSATS simulation is based on a conceptual model derived to meet the needs of a message-based simulation. The model, shown in Figure 1, is defined in terms of input events and output events (messages) and/or state changes. The generic actions shown, when taken as a whole, represent the interpretive logic inherent within each nodal model. These logic actions execute serially in response to the input event received.

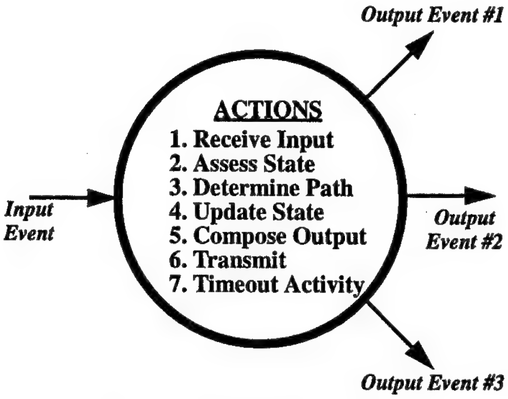


Figure 1: The Nodal Logic Model

When an input message is received, the data within the input message and the current mission state contained in the tactical state tables is compared against a rule set to determine the single logical pathway to be followed. When a specific path is selected, a series of response actions is instantiated to perform state table updates and to generate output messages. When the response actions are completed, the nodal simulation will enter a state waiting for its next expected event. Each pathway may have a timeout activity associated with it, which is used to perform file cleanup and activities to continue a mission thread if, for some reason, an expected input event does not occur.

Extreme care must be taken when defining the logic contained within the Nodal Logic Models. As C^2 systems move into the realm of more sophisticated communications and data formatting, consideration must be given to the data preprocessing which occurs in the systems being simulated.

The Nodal Logic Model is deterministic in nature, which is essential to effect a suitable simulation which can be implemented in software. Nodal simulations must behave in a predictable manner to ensure exercise repeatability and to properly assess the behavior of the live nodes which make up the SUT. Random events are not precluded from the Nodal Logic Model but must be carefully integrated into the rule set which is used to determine response pathways. In this manner, random events may be considered without destroying the overall deterministic character of the basic model.

4. Design and Implementation Strategy

The development of FSATS is progressing in parallel with the development of the C2 systems it is intended to support. The emerging design of these systems, as well as user requirements, has driven some key design issues which add benefit and robustness to the resulting simulation.

4.1 General Design Strategy

The FSATS requirements for a reusable and distributed simulation were key elements in the decision to utilize a modular, object-oriented approach to the design. Each nodal simulation is defined as a specific object and can be allocated to and de-allocated from specific processors at will. The interfaces between these simulation objects mirror the tactical messages which would be sent between their respective live nodes.

The current FSATS simulation was implemented using data-defined logic records which are manipulated by a runtime interpreter. This decision minimizes the impact of changes brought about by the emerging design of the supported C2 system and the emerging logic models themselves. The design is therefore easily adaptable to other applications and to future enhancements. Several enhancements have already been identified to allow this simulation to meet other needs, such as training.

The deterministic nature of the nodal logic models lends itself to a reimplementation of this simulation using a rule-based expert system. Hence, a design was incorporated which would minimize the workload necessary to transition to this technology. This design enhancement is currently under consideration.

Since the simulation consists of many functions which are repetitive in nature, such as extracting data from messages and using it to update the tactical state database, FSATS has emphasized the use of reusable software components called common actions. These common actions are highly modular, which supports the maintainability of the simulation.

4.2 Maintenance of the State Information Database

FSATS operates as a flexible, distributed simulation which allows the assignment of simulated objects to the various FSATS processors. All simulated objects within a given processor have access to two types of database tables used to support the simulation. A master set of tables contain data (normally static) applicable to all simulated objects. A second set of tables contain the dynamic tactical state information used to affect the logical behavior of the simulation models and populate tactical messages.

The current FSATS implementation makes a unique set of the master tables available to each processor. Changes to these tables are currently limited to those performed directly by the FSATS operator during planning or during pauses in the execution of the scenario during runtime. These limitations are necessary due to a lack of any suitable data exchange media to provide either a pathway for the distributed processors to gain access to a single master database or a pathway to accurately update any distributed set. The tactical communications paths cannot be used for this purpose, since the system requirements dictate that FSATS not alter the tactical performance of any node or network. Future versions of FSATS may incorporate an instrumentation network, which will provide a suitable means of manipulating a distributed database.

The process used to update the dynamic state tables must be carefully defined and controlled. Although distributed simulations normally share database information, this concept of global access is not applicable to an effective C2 simulation. Since multiple nodes on a given processor can see the data within these dynamic tables, the access must be carefully controlled to ensure that a state update performed by one node cannot be used by another, prior to that node receiving a suitable tactical message which would result in that update taking place in a normal fashion.

For example, if Node A conducts a tactical move and updates its location within the dynamic tables, Node B should not be able to access that new location until a message is received which indicates to Node B that Node A is in a new location. This phenomena has required FSATS to maintain state information peculiar to individual nodes, as well as other information which represents "ground truth." For reasons similar to those discussed for the master tables, the ground truth information will only be accurate within the data structure which resides on one specific processor. The instrumentation network will provide a solution to this problem as well.

4.3 Event-Driven Reactive Simulation

The nodal logic which forms the basis of FSATS was created to produce a reactive type simulation, which will respond in a manner suitable to the current state of node being simulated and the nature of the input event it is subjected to. This feature of FSATS makes it possible for the simulation to support a given exercise without the need for operator intervention. Scripted events in the form of a Time-Ordered Event List (TOEL) may be used by a simulated node to initiate selected missions, but no operator decisions or actions are required to adequately respond to the input messages received by the simulation. The TOEL events merely represent a human decision to identify and initiate a mission which the overall system need perform, and to provide a entry route of the mission-related data into the system to replace the keyboard entry normally performed by the operator.

4.4 Time Synchronization

Although FSATS operates principally as an event-driven simulation, the overall simulation functions within a scenario specified by artificial time limits. Also, a significant number of the mission threads require a knowledge of the time in which the controlling nodes operate. Hence, it is imperative for the distributed simulated nodes to gain and maintain a single concept of time.

FSATS utilizes the Global Positioning System (GPS) to maintain time synchronization within the simulated nodes within a single millisecond accuracy. The Universal Time Coordinated (UTC) provided by GPS is available to all FSATS processors, since each target system contains an antenna and the necessary GPS hardware. An offset value is calculated to convert this UTC into a valid scenario time, which is available to all simulated objects.

4.5 Simulation Control

While operator intervention is not desirable to maintain the simulation's progress, there remains a critical need for the FSATS user to be able to control the execution of the exercise being supported. These needs include the ability to control the pace of the exercise, as well as pausing, stopping, and resuming the exercise. However, the simulation objects must operate exclusively in real time.

The reactive nature of the simulation models and the sensitivity of the tactical messages with regard to scenario time necessitated isolating the software object which controls the injection of the TOEL messages into the scenario. This software object, known as the TOEL_Server, may be altered to vary the speed (as a function of percentage of real time) that the messages required to initiate missions are injected into the simulated objects. This allows the user to control the rate of scenario execution without affecting the real-time performance of the simulated nodes. Additionally, the TOEL_server (as the object is known) may be halted independently, which allows the simulated nodes to continue to respond to the tactical messages generated by the live node until all mission threads have been terminated in a normal fashion.

4.6 Simulation Abstraction

As the size of the message domain grows - with the addition of new systems, formatted messages, and communications protocols, - the simulation must be able to operate properly in an increasingly complex dynamic environment. A key to success has been in maintaining the concept of simulation abstraction, as shown in Figure 2. This abstraction allows other software components to normalize the incoming tactical message into an internal data form which may be utilized by the simulation software. In this manner, the simulation is more adaptable to changes in tactical messaging and to integration with other systems, and it is far more easily maintained.

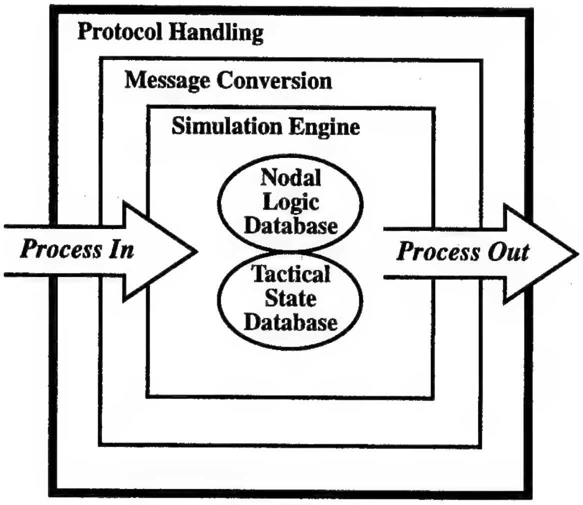


Figure 2: Simulation Abstraction

When an incoming message is received by FSATS, a software component known as the Tactical System Interface (TSI) performs the routine protocol handling. The contents of the tactical message are then sent to the Tactical Message Translator (TMT) which converts the bit stream received into data types which can be used by the simulation. Additionally, since FSATS deals with multiple message formats, data sets, and message handling techniques, the data is normalized to store the state information in a single common form.

The simulation engine provides additional abstraction for the simulation, as well as promoting software reuse and modularity. The simulation engine consists of two principal parts. The State Manager controls access to and performs the read/write operations required of the Tactical State Database. The Runtime Interpreter uses data from the input message and from the state manager to interpret the rule set contained within the Nodal Logic Database. The common action procedures are then sequentially instantiated to update state and to build output messages.

5. Additional Applications

Although FSATS is being developed to support C^2 testing, this type simulation can be adapted to meet other needs. Any changes required to meet the needs of other areas may normally be addressed with a minimum of effort. Some possible applications which are currently under investigation include the following.

5.3 Collective Training

There is a direct application to the needs of collective training, since the command post training (CPX) or field training exercise (FTX) environments are identical to those used in operational testing. Normally, little adaptation would be required to support training, since it is of benefit to produce doctrinally correct logic models, which contain a very detailed rule set to avoid the appearance of errors in the tactical messages produced.

5.2 Doctrinal Analysis

To explore the impacts of doctrinal changes prior to implementing these changes within the total force, full simulations using nodal logic can be run with a variety of input events and exercise configurations. Also, areas such as bottlenecks, information gaps, and network overloading could be identified.

5.3 Interoperability

Since the FSATS simulation is defined in terms of input and output messages, the internal logic could be simply defined to translate the input data into another form. In this way, a simulation of this type could be used to affect an interface between two existing systems which currently lack such an interface.

Distributed Interactive Simulation for Intelligence Data Dissemination

F. D. Magee
P.O. Box 4232, 34645

Abstract

This paper is conceptual in nature, discussing the application of Distributred Interactive Simulation (DIS) technology to the dissemination of strategic and tactical intelligence information to a broad base of military tactical decision makers and senior leaders. While the current thrust of DIS development is oriented toward the support of apriori training, systems acquisition engineering analysis and mission rehearsal, the proposed concept advocates DIS as primarily a real operations decision-making support tool with mission-concurrent training and incremental mission rehearsal support capability.

Introduction

The view that "the problems endemic to military intelligence did not arise overnight, and their complexity defies an instant solution"[1] has recently been echoed in professional military publications. Maj. Raymond J. Leach in favoring review of Maj. James P. Marshall's recent book, Near Real Time Intelligence On The Tactical Battlefield, "...urges a return or reprioritizing of our efforts to tactical intelligence support, and suggests that much of our standard doctrine is outmoded... Perishable combat intelligence of immediate tactical value should not be restricted...,but pushed down and laterally, using the latest technology advances."[1] In this era of anticipated small unit low intesity conflicts and multi-national force operations, time critical intelligence data dissemination to the small unit tactical decision maker becomes or paramount importance. Also the burden of intelligence data integration falls heavily on the tactical decision maker. Textual reports with varying time stamps require time and the commander's undivided attention to piece together.

The military "...flies a number of "national level" collection platforms without receiving a payoff in [effectively disseminated] tactical information."[1] Currently, as this raw intelligence data from electro-optical sensor sources traverses the many gates of filters and enhancement processes, it ceases to be the pure reflection of the environment it depicts. The enhancements are necessary because to the untrained eye the raw imagery may not be recognizable for its true or maximum intelligence worth. "Making sure the imagery gets to the right analyst and with sufficient time to allow commanders to take advantage of the data to reposition their forces is far easier to envision than to incorporate in a functioning system that can be fielded."[4]

Within the may pixels required to represent an intelligence image, a subset serve as cues which lead to real inteligence information. The other pixels may serve to indicate other non-intelligence related information and a third class of pixels are virtually worthless from an intelligence standpoint. Current dissemination schemes compress and digitize entire images, then send them across the networks to secondary users and sometimes to the ultimate users. If the raw imagery is considere to be a pure reflection of the environment that it depicts, then after the enhancements and compressions it has moved more toward a simulation grade of reality representation.

0-8186-6440-1/94 $04.00 © 1994 IEEE

Today's military application for synthetic environments is mainly oriented toward training. The air elements have flight training systems and the ground elements have tank trainers. Soon, planners will be engineering ways to place the DIS synthetic environment systems on field operational platforms for training purposes. A more comprehensive approach is needed which integrates the training value of these systems with their at least as equally valuable intelligence dissemination and mission rehearsal potentials.

This paper outlines the operational and architectural concepts for applying DIS to the dissemination of time-critical intelligence information to the small unit commander.

Operational Concept

Overview

As image generator computers come down in size and price, it will become increasingly more practical to place a dynamically networked image generation and display system with the small unit tactical commander. With free viewpoint movement within a limited but large enough area of interest, the commander will be able to make synthetic reconnaissance and see discernible representations of critical intelligence information. Threat entity states will be updated in near real time. For instance, a helicopter firing a missile at a tank, and the tank suddenly changing appearance reflecting the hit, will be a viewable scene for the commander almost immediately following the real world event. For small unit commander too busy to integrate and assimilate the floods of relevant intelligence data, the intelligence DIS Field Node will allow the commander to address the problems at hand while keeping one eye on the 3-D simulation of the current situation. Friendly and threat entity states will be updated in near real time. Changes in the terrain and fixed cultural features will change periodically to reflect reality. For example, a bridge may at one moment appear serviceable and at the next moment be destroyed. Changing weather and trafficability conditions will be incorporated. This DIS intelligence dissemination concept involves a system comprised of five segments; the Central Data Base Generation Segment (DBGS), the Theater Data Base Generation Segment, the Transmission Segment, the Field Node Segment, and the Instrumented Field Entities Segment. Multiple types of data will flow across the networks serving the DIS field nodes; environment database updates, threat entity states, friendly entity states, aggregated entity states. Voice communication will be the responsibility of other systems.

In addition to the 3-D free viewpoint image of the tactical area of interest, the decision maker will have 2-D sub-window overview of the area with iconic entity/featural representations. Much of the non-pictoral intelligence data related to the displayed icons will be accessible in a windows/menu style user interface environment. For instance, a bridge icon on the 2-D overview display may be selectable and information such as the serviceability status of the bridge, its loading capacity and the freshness of the supporting intelligence data, will be displayed. While accessing this supplemental information, the commander will be able to continue viewing the tactical situation on the 3-D display.

Central Data Base Generation Segment

National intelligence collection systems, collect data, some of which is valuable to the tactical decision maker. This data may be fused with tactically collected intelligence by the tactical fusion systems. However, terrain and cultural environment data and national weather agency data will augment the data from these national intelligence sources to create the initial visual environmental data base. Later this baseline will be used as the basis for the overlay of tactically collected and fused data at the theater level (Here, "theater" means the broad area of tactical operations or interest).

The basic databases will be generated in advance of unit deployment. It will be important to streamline the current semi-automated processes for developing these visual databases because many of the coming crisses will ill-afford more than a day or two to prepare for deployment. This basic DIS visual environment database will be stored for transport on compact discettes. Over prolonged periods of deployment the tactical commanders will receive update database discettes via the normal physical distribution used for other controlled classified materials.

An example of nationally collected intelligence data is imagery taken from a reconnaissance satellite of a POW compound which for specal operations purposes must be modelled and fixed into the environmental data base. The Central Data Base Generation Segment would send this environmental data base to the Theater Data Base Generation Segment for further processing.

Tactical intelligence collected from theater level collection assets will be funnelled into tactical fusion systems such as the Army's ASAS. "The [All Source Analysis System] ASAS is the Army's portion of a former joint effort with the Air Force known as the Joint

Tactical Fusion Program. The initiative is designed to gather data from all available sources, including electronic intelligence (ELINT), signal intelligence (SIGINT), imagery intelligence (IMINT), human intelligence (HUMINT). fuse that data with on-going combat information...". [4] This fused data will overlay the database provided and updated periodically by the Central Data Base Generation Segment.

Theater Data Base Generation Segment

At the Theater Data Base Generation Segment, fusion processes further interpret the imint/photint.data. Fusion systems integrate them and output products more conducive to object-orientaiton. Object-orientation of the intelligence data is critical to the concept of using "object-oriented environmental servers" to create the basic common picture for the tactical comanders. Recently, there has been research done to pave the way for object-oriented environmental servers which, directed by the DIS community, will serve the DIS intelligence community well.[5] Object-orientation allows different types of intelligence data related to the same object/entity and coming from vaious sources may be collected and displayed at the DIS Field Node in multiple forms. Both threat objects and friendly objects will be treated as "persistent objects"[6].

The theater level database generation process may be partitioned into processes which may be automated in near-real-time and non-automated labor intensive processes. The building of visual databases, which provide for geographically specific depiction of the environment and sometimes includes high resolution imagery specific texturing of models and features, is known to be a long labor intensive task for even small areas. The automization of this process will require partitioning and parallelization of database generation tasks. Those tasks which are inescapably labor intensive such as detailed model building will be accomplished off line. No doubt, there will continue to be improvements to the model-building tools in this area. Continuing with an earlier example, a tactical remotely piloted vehicle collects more imagery of the compound. The Theater Data Base Generation Segment will take this database region modelled at the Central Data Base Generation Segment and update the model, providing the scene densities needed for a special operations rescue mission.

The Theater Data Base Generation Segment will take the outputs of the multi-source intelligence fusion system, partition the data into entities and environment classes. The environmental data base received from the Central Data Base Generation Segment will be rebuilt integrating the environment class of filtered intelligence data. For example, an intelligence report indicates a direct hit of a missile on a bridge making it unservicable. A damaged bridge model will be substituted for the healthy bridge. Dynamic simulation of weather effects will be treated as universal features or moving models. While the weather environment will be included in the data base from the Central Data Base Generation Segment. The alteration of the baseline environmental database will occur at the Theater Data Base Generation Segment

Recompiling of the database will be accomplished decentrally at the DIS Field Nodes so that only DBGS software change instructions will be sent in order to effect an update. In this way minimal communications bandwidth will be required to support environmental database updates. There will be a controlled sequencing of these changes so that DIS nodes will process changes in chronological order, presenting scenes to the commander which are held in common by adjacent field nodes.

Transmission Segment

The Transmission Segment function is to provide communications, application layer through physical layer, to effectively support the DIS intelligence dissemination concept. Entity data (usually threats) will go a separate route eventuating in DIS PDUs directed over the DIS net to selective subnets. The DIS communications network will optimize data flow down from the central and theater DBGS centers to the Field Nodes while minimizing the broadcast loads from the the Field Nodes.

Only entity/environment data which changes will be communicated. There will be a need for unambiguous rules of entity data control to preclude inconsistent representations of entities while maximizing dissemination of time critical data. The "Persistent Object Protocol"[6], developed for interfacing semi-automated force (SAF) systems, will provide a methodology for intelligence DIS entity information control. Ownership of an entity/object may change but there will be well-defined responsibility for updating information of each entity on a periodic basis.

Field Node Segment

The Field Node Segment will be the hub of all DIS intelligence data. Intelligence DIS data will be received by the Field Nodes from the Theater Data Base

Generation Segment, the Instrumented Field Entities, other Field Nodes and occassionally directly from the Central Data Base Generation Segment. While strategic intelligence raw data will not traverse the DIS network, pointers which relate DIS depicted intelligence data to its raw source(s) will be forwarded within the DIS PDU structures. These pointers will allow the commander to verify what is depicted on the 3-D or 2-D display, given the availability the raw data at the DIS field node received from the other intelligence dissemination systems. Most likely, for some time, the availability of raw strategic imagery to include radar and IR will be confined to the medium to larger size units, due to communications and security constraints. Larger units also will have DIS Field Nodes, but will receive the annotated enhanced imagery products, compartmentally disseminated via other systems.

DIS entities will be aggregated by the DIS Field Node for higher reporting purposes and for the benefit of senior tactical commanders having DIS nodes. Without filtering, the 2-D overview display will become cluttered with icons. While the senior commmander will definitely want the capability to adjust the DIS node communications filters to examine closely a small unit commander's situation, the main emphasis at this level will be management of the bigger picture which will require aggregated entity status. The unit status reporting function of the DIS Field Node wil be of lower priority than its primary function of downward intelligence dissemination.

Instrumented Field Entities Segment

Local instrumented entity inputs to the DIS Field Node will be carried via established tactical digital data networks. The Field Node Segment will take these DIS data packets and filter them and integrate them into the current tactical picture. This host will drive the IG, activating/deactivating models, repositioning models which represent entities, and reload area of the environmental database into the image generator to show updated terrain and cultural features in the area of interest to the commander.

Also, the DIS Field Node will form DIS Protocol Data Packets (PDUs) to output onto the DIS network the locally known entity state information.

Local instrumented entities will be reporting back to the small unit commander in PDU packets their positions, health, logistical states automatically with programmed periodicities. The DIS Field Node image generator controller will send the position changes to the image generator in near-real time. Emphasis will be on automated intelligence gathering and reporting between the local friendly entities and the DIS field node, and between DIS field nodes.

Consider the following scenario. A tank platoon commander DIS Field Node receives Global Positioning Satellite (GPS) sourced entity state PDUs from a close air support DIS instrumented entity. The tank commander after seeing the simulated battlespace from the supporting pilot's viewpoint determines that the pilot's position in deep defilade is insufficiently supportive. Meanwhile the co-pilot of the supporting aircraft uses a DIS Field Node to make reconnaissance of terrain features forward of the tank platoon's position, to plan for optimal support movement. DIS entity state traffic illustrates for thet co-pilot and tank platoon of friendly multi-national force T-72s has moved into position just over the next ridge, out of line of sight communications. The T-72 positions had arrived via the tactical intelligence DIS network.

Field training and mission rehearsal

While the primary focus here is to discuss the concept and effects of employing DIS technology for intelligence data dissemination. The current mainstream of DIS development focuses on training, systems acquisition engineering analysis, and mission rehearsal appplications, all of which are consistent with intelligence DIS concept. For training purposes fictional intelligence data or scenarios will be served onto the training exercise DIS network consistent with the scenarios and visual scenes presented through the virtual reality manned training simulators. DIS standardization will make this possible.

During real operations when the activity is in a lull, constructive simulations local to the DIS Field Node will feed tailorable scenarios to the tactical planner. A virtual "what if" game will be played iteratively, as time permits. These scenarios may involve real entities or may be confined to the DIS Field Node for analytical purposes only. Decision support software in tandem with the DIS Field Node representation of fictitious tactical scenarios will help prepare the tactical decision maker for the otherwise unexpected. In the not to distant future, the system user will use voice activated computer commands to change viewing parametric values such as zoom or changing viewpoint.

Architectural Concepts

Central DBGS

The strategic level DIS automated DBGS will be a large processing center in CONUS which will employ in the near term high bandwidth long lines communications to update the DIS environment baseline at the Theater Data Base Generation Segment. From this center, pre-deployment visual databases will be prepared, packaged and distributed. This center will be directly associated with the national level mapping and intelligence collection resources. While the source data for the DIS environment will be controlled at TOP SECRET compartmented security levels, the output database product will be controlled colaterally at the SECRET level. No characteristic of the output database will reveal the sources identities or resolution capabilities from which it may be derived.

Theater DBGS

There will be a theater level DIS automated DBGS (theater central environment server) which will be a shelter sized processing center. The tactical communications networks indigineous to the hosting unit will support the DIS traffic. This will involve multiple types of communications (SHF, UHF, HF, commercial lines) and multiple link pathways with redundancy. The intelligence DIS system dissemination timeliness will be a graceful function of Transmission Segment service. Entity type prioritization will help optimize DIS bandwidth utilization.

DIS Field Node

From a users's perspective, the intelligence data dissemination DIS network would be structured as follows. In the near term, i.e. 1990's, a desktop computer will serve as thecommunications front-end processor and the image generator controller. Also, this computer will manage non-visual intelligence data and host constructive simulation software, mission rehearsal and incremental training scenario generation software. The IG will require another box. It is basically a special purpose computer composed of a control processor, geometry processor and a pixel processor. Within the last few years, even the high-end real-time IG's have shrunk from a row of well-populated 72" high cabinets to a small free standing box. With reasonable confidence one could speculate that by the year 2000 the high-end image generator will fit easily on a desk top, and arguably to small man-pack size by the year 2010.

The display will be a color monitor and in the near future, by year 2005, be integrated onto the already marketed helmet mounted display type. 3-D visual free play on the image generator will be directed through a joystick control. And the current industry standard windows type interfacing environment will apply to the activation of the 2-D overview subwindow and intelligence data pull-down sub-windows.

In the near future DIS Field Nodes will be present on battle tanks and aircraft, where GPS will be the primary source of DIS entity position status. In the far term, individual personnel position and status on the local DIS subnet will be a reality.

Communications front end

The communications interface at one end connects to the communications carrier, military or other. In translates the signal received from the local carrier across the various layers of protocol (ISO) up to the application layer. The application layer protocol, known as the DIS protocol is a currently developing data packaging structure which will serve as an excellent baseline for service to the intelligence DIS dissemination mission.

Filtering

Incoming DIS PDUs will be filtered, allowing only the PDUs pertinent to the particular DIS Field Node to be further processed. The majority of these will be Entity State PDUs. The environment database change PDUs will be less frequent and more lengthy, but not a significant drain on the communication resources.

Entity states data received will be adjusted relative to the age of the data calculated from the time stamp which will accompany the position, orientation and other data. This adjustment, called "dead reckoning" is an extrapolation of entity position (latitude, longitude, altitude) and orientation (roll, pitch, heading). In DIS exercises involving manned simulators, it is sometimes desireable to smooth entity position and orientation corrections. With Intelligence DIS, data nearest to reality is welcome even if it means abrupt change in the scenes. Other range filters may partition the remaining group of dead reckoned entities into those viewable given the currently loaded environmental database from those not immediately viewable but near enough in the vicinity that they may be viewed and ought to be tracked. Generally, other processing of this type which will take place in the DIS Field Node will be executed in an effort to off-load critical processes or accomodate the constraints of the image generator.

Transformation of the visual data base

The DIS Field Node will also serve as a local Data Base Generation System(DBGS) for environmental database updates which arrive via DIS channels. The local-DBGS process will be fully automated and will be similar to one of the automated data base generation process architectures in operation at the Theater Data Base Generation Segment. The list of data base transformation instructions successfully executed to update the visual data base at the Theater level will also be executed at the Field Node level to maintain visual data base baseline comonality. Near real time updates make at the Theater Data Base generation Segment to the visual environment databse will require databse transformation instructions. These data base transform instructions will be communicated to and repeated by the local-Data Base Generation System to bring up the local baseline visual database. This will work for changes to an existing visula data base. New areas will require physical distribution or lengthy file transfers, given high bandwidth communications availability.

The visual database will be formatted locally for use with the image generator. Because, only portions of the Visual Data Base which will be affected by the changes will be restructured, the time it will take to reformat changes will be minimal. Of all the DIS Field Node processes, the local-DBGS will be the most time consuming, but will be independent of the time critical entity state update process.

Non-visual intelligence data management

Other data associated the entities representable by models on the DIS Field Node display, will be best displayed in the form of text or symbols. The DIS Field Node will manage this data and buffer it ready for display in 3-D. Some symbols will be displayed within the 3-D scene. The majority of this data will require a button selection on a display window. The 2-D display sub-window will depict information symbolically with some alphnumerics. However, this display with its icons depicting entities in the view area will get cluttered. The DIS Field Node will offer 2-D and 3-D loading parameters which may be set by the operator to optimize information display.

Image generator control

In order to dynamically change the simulated environment terrain, sea state and cultural features without interruption of service to the commander, a two channel image generation system will be employed with only one channel displayed at a time (i.e.ping-pong). This will give the IG a chance to load changes to the environment off-line. The channel inter-change cycle period will be approximately 5 minutes, given the local database recompilation function is performed prior to loading. Moving model sets will be loaded in similar fashion, off-line. Activation of multiple instances of a loaded moving model type will be almost instantaneous, within the image generator's scene loading constraint.

Operations and display

In the near term, two display CRTs will be necessary for DIS Field Node operation; an operator terminal and one for 3-D high resolution display. A joystick will interface with the image generator allowing the commander to freely reposition and reorient the 3-D viewpoint on the 3-D display. A standard keyboard interface with a mouse will allow the commander to select windows and Field Node options on a standard desktop computer terminal. In the near future all operations and display features both visual and physical will be resident in one box. In the more distant future (around year 2010) these functions will be supplied through the Field Node back-pack and helmet display.

DIS instrumented field entities

The automated status from local networked Instrumented Field Entities will follow the standard DIS protocol rules now in development. These entites may be tanks, planes, personnel etc. and may be outfitted with a variety of instruments or could be DIS Field Nodes themselves. With the automation of these field instruments the field operator will remain free to concentrate on mission objectives.

Status of threat entity positions/states will, through voice recognition systems, be automatically relayed to the DIS Field Node [My thanks goes to Dr. Richard Economy, Martin Marietta Corporation, Daytona Beach, for sharing this concept with me].

Areas for Advancement

The DIS concept for intelligence data dissemination discussed above requires technology advancements in three areas:

1. The translation of fused tactical intelligence into entity or featural object-oriented data; an integration task probably requiring AI methods.
2. The partitioning of the Environmental

Data Base Generation processes; streamling the labor-intensive off-line tasks, and automating the near-real time on-line tasks.

3. Data Base Generation System software techniques which allow efficient compression of the Environmental Data Base transformation instructions executable by the DIS Field Node's local data base generation function.

Summary

There is a need for more effective dissemination of strategically and tactically collected intelligence to the small unit tactical decision maker. The simulation community in their efforts to standardize Distributed Interactive Simulation protocols and practices for training and acquisition analysis purposes have also begun to lay the foundation for the application of DIS to the dissemination of tactical intelligence information. The DIS Field Node will allow the commander to quickly assimilate fused intelligence as it is integrated in near-real-time onto 3-D joystick geographically specific display and corresponding 2-D overview map display environment.

The flow of data to the small unit commander starts at the strategic level, Central Data Base Generation System. The tactical level, Theater Data Base Generation System converts updated fused intelligence into an updated visual database, building onto what the central level has established. From this tactical center, environmental data base update instructions and entity stat PDUs flow to the DIS Field Nodes.

The DIS Field Nodes receive, process, and display the simulated operational scenery. The commander will view the areas of interest and pull up associated supplemental and verification data to increase operational awareness and intelligence confidence, respectively. The DIS Field Node will receive locally generated entity state data, via PDUs. This data will be transmitted from locally networked DIS-instrumented field entities. Aggregation of local entities states will permit status reporting from the Field Nodes upward to the theater level. In the near-term a DIS Field Node architecture will require three terminal-size boxes. With ASIC technology advancement and the successful integration of helmet displays, the DIS Field Node will reduce to man-pack size.

References

[1] Raymond J. Leach, Maj. USMC, "The Information War", Marine Corps Gazette, Marine Corps Association, Quantico, VA, September 1994, p. 107.

[2] James P. Marshall, Maj. USAF, Near Real Time Intelligence On The Tactical Battlefield, Air University Press, Maxwell Air Force Base, AL, 1994.

[3] Raymond J. Leach, Maj. USMC, "Information Support to U.N. Forces", Marine Corps Gazette, Marine Corps Association, Quantico, VA, September 1994, p. 49.

[4] Mark Tapscott, "New Pictures Emerging in Battlefield Intelligence", Defense Electronics, April 1993, p. 31, 32.

[5] William H. Horan, Michael J. Smith, Curtis R. Lisle, Marty Altman, "An Object-Oriented Environment Server for DIS", Institute For Simulation and Training, University Of Central Florida, Summary Report, 9th Workshop on Standards for the Interoperability of Defense Simulators, Volume I, September 1993, p. A-95.

[6] Joshua E. Smith, Anthony J. Courtemanche, Richard Schaffer, "The Persistent Object Protocol", Institute For Simulation and Training, University Of Central Florida, Summary Report, 9th Workshop on Standards for the Interoperability of Defense Simulators, Volume I, September 1993, p. A-199.

Session 2A:

DEVS Formalism: Simulation Engines and Performance Modeling

Distributed Simulation of DEVS-Based Multiformalism Models *

Herbert Praehofer and Gernot Reisinger

Institute of Systems Science
Systems Theory and Information Engineering
Johannes Kepler University Linz
A-4040 Linz, Austria

Abstract

In this paper we introduce a new approach for parallel, distributed simulation of modular, hierarchical DEVS and DEVS-based combined discrete/continuous multiformalism models. The algorithm combines conservative and optimistic distributed simulation strategies and is able to optimally exploit lookahead capabilities of the model. The object oriented implementation in C++ is intended to serve as a powerful simulator in the STIMS modeling and simulation environment.

1 Introduction and Motivation

The DEVS modeling and simulation [23] approach is an attractive alternative to conventional message-based modeling approaches used in distributed simulation. As expressive as any other discrete event modeling paradigm, it serves it merits by its set-theoretical basis, its independence of any computer language implementation, its independence of any particular field of application, its modular, hierarchical modeling methodology, and its clear system theoretical terminology. In DEVS modeling, complex models are built by coupling together atomic building blocks, i.e., connecting the ports of well defined input and output interfaces. Models can be built in a hierarchical manner, i.e., coupled models again can serve as components in more complex coupled models. Similar to finite state automaton, atomic DEVS models' dynamic behavior is defined by state sets and state transition and output functions. DEVS distinguishes two type of events - *internal events* are time scheduled and handled by the *internal transition function*, *external events* occur upon the arrival of inputs at the input ports and are handled by the *external transition function*. We base our research on the DEVS approach to discrete event modeling and simulation.

Several implementations of DEVS-based modeling and simulation concepts have been done [23, 21, 18, 11]. The STIMS modeling and simulation environment [18] is a CommonLisp based general purpose environment also allowing combined discrete/continuous modelling and simulation.The DEVS extension to combined discrete/continuous modeling introduced in [16] provides system theoretic modeling formalisms for modular, hierarchical combined modeling. STIMS is a fully integrated, interactive environment organized into several layers which are targeted to model development, simulation execution, and simulation data analysis. It provides an approach for visual interactive specification of atomic and coupled DEVS-based models [19].

In the research project presented here we will realize an object oriented implementation of DEVS-based models in C++ and a distributed simulation protocol. We introduce a new distributed simulation algorithm for DEVS-based models and discuss its object oriented implementation. The algorithm combines conservative and optimistic strategies. Simulation processes work conservatively as long as this is possible but will continue with a riskfree optimistic scheme afterwards. The scheme is able to optimally exploit lookahead capabilities of the model by computing accurate estimates for the earliest input to be received at a component's input ports. It also allows the distributed simulation of components which have continuous internal behavior but interact by events only. The C++ model implementation and simulation protocol is intended to serve as a powerful simulator in the STIMS modeling and simulation environment.

The simulation protocol has been design with the following objectives in mind:

- The distributed simulation algorithm should show good performance on multiprocessor machines with a low to medium number of high power processor nodes, like clusters of workstations or shared memory multiprocessors. Such multiprocessor architectures are widely available and therefore are suitable for a general purpose simulation environment.

*work supported by Austrian Science Foundation FWF

0-8186-6440-1/94 $04.00 © 1994 IEEE

- Distributed simulation should require minimum additional coding from the user. At the current version of our system, the user has to do the mapping of components to processes, has to implement methods to pack and unpack objects of user defined types, and has to provide information about lookahead characteristics of the models.
- As our scheme supports combined discrete/ continuous modelling and simulation, also the distributed simulation algorithm should support it. Numerical integration of components which only interact by discrete events can be done in parallel.

In the following we first describe our new approach to distributed simulation of DEVS-based models. Then we compare our approach with other approaches to distributed simulation. Finally we discuss several issues of the object oriented implementation in C++.

2 A New Approach for Parallel Simulation of DEVS-Based Models

In the following we introduce a new approach for parallel, distributed simulation of modular, hierarchical DEVS-based models. For distributed simulation of DEVS-based models, the hierarchical structure is flattened by resolving the hierarchical coupling structure to direct couplings of atomic components. Then the atomic components are divided into several clusters where each cluster is simulated by one parallel simulator process running on one distinct processor.

Our approach is based on the idea to combine sequential, parallel conservative, and parallel optimistic event processing. As long as possible, i.e., as long as it can be guaranteed that no inputs will arrive from other processes with a time earlier than the local event time, the parallel simulation process can process the events in sequential order. Then, it will try to exploit lookahead capabilities in its local components to continue processing events which are safe. Finally, when neither sequential nor conservative event processing is possible, it will continue to process events optimistically. However, optimistic event processing is done riskfree, i.e., outputs to other processes are not sent and in this way effects of optimistic event processing are kept local to the process.

Crucial to the scheme are input time estimates which for every input port of atomic components give lower bounds of the next input to be received at that port. These time estimates are computed by the influencing components' output time estimates and serve various purposes. So they are used for global as well as for local synchronization. First, the time estimates of the inputs coming from other processes are used to determine the time until when sequential event processing can be done. Second, the input time estimates determine the time for each atomic component until when event processing is safe. Third, the time estimates are used in optimistic event processing to manage fossil collection. In the following we will describe the approach in detail.

2.1 Computing Input Time Estimates

Time estimates $eit_{j,ip}$ of the earliest input to be received at the input ports ip of atomic components j are computed taking the minimum of output time estimates of the output ports op of components i coupled to ip. Figure 1 gives a cluster of four components simulated by one process p together with its interprocessor couplings. The minimum of earliest input time estimates

$$eit_j = \min_{ip}\{eit_{j,ip}\}$$

for all input ports ip of one component j gives a lower bound for the next input to be received by component j. Additionally, the external input time estimate $xeit_p$ giving the earliest input to be received by any component of process p can be computed by taking the minimum over the inputs coming from components of other processes. Obviously, an event in a component j with event time t smaller than eit_j is safe to process. Additionally, a simulation process can process all events whose event time is smaller than the external input time estimate $xeit_p$. The scheme has proven to avoid causality violations [17].

The output times estimates $eot_{i,op}$ for component i are computed exploiting lookahead capabilities of the model. In case only static lookahead [14], i.e., minimum time delays $d_{i,ip,op}$ between input at port ip and outputs at port op of components i, are known, the output times for a particular output port op can be derived by

$$eot_{i,op} = min\{tn_i, \min_{ip}\{eit_{i,ip} + d_{i,ip,op}\}\}$$

where tn_i is the time of the next internal event. For a source component, i.e., a component s without an input, we define the eit_s being equal to infinity. For such a scheme of computation of the output times we proved that the simulation does not deadlock under the constraint that in every feedback loop there is at least one component with a minimum time delay $d_{i,ip,op}$ strictly greater than zero [17].

Dynamic lookahead [14] is lookahead computed based on the current state of the component. We can compute better time estimates $eot_{i,op}$ for an output at port op by

$$eot_{i,op} = lookahead(s, tn_i, eit_i)$$

where *lookahead* is an arbitrary complex function to compute the dynamic lookahead for output port op.

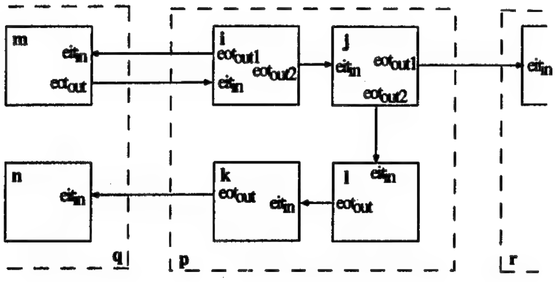


Figure 1: Times maintained for coupled DEVS model.

Obviously, if at least one of the components uses a minimum time delay strictly positive in the computation of its dynamic lookahead, also this scheme is free of deadlock.

2.2 Local Event Processing

Event processing on one simulator process p is done in the three stages as described above: first events are processed sequentially, then conservatively exploiting lookahead, and finally optimistically. The simulator process accomplishes its task using the earliest input time estimates and a list of event times tn_i for its components. The pseudocode below depicts the event loop for a simulation process. In the indefinite loop it is tested if the minimum event time tn_{i*} is smaller than the time estimate $xeit_p$ giving a lower bound of the next input received from a different simulation process. If so, this event can be processed and all its resulting outputs can be processed immediately. If not, then event processing might not be safe because inputs from other processes might cause causality violations. However, lookahead can be used to determine other components which are safe. The next event in component i with event time tn_i is safe to be processed if it has an earliest input time estimate eit_i greater than tn_i. This event can be processed. Resulting outputs are distributed to the influenced components and are processed immediately if they are safe. Otherwise, they are inserted into the event list. In case that no safe event can be found exploiting lookahead, events can be processed optimistically. The component with minimum event time is selected to do so. Resulting outputs are also processed optimistically for the local components but are not distributed to other processes. This means that eventual rollbacks are kept local to the process and therefore are not too expensive, i.e., the scheme is riskfree optimistic.

```
loop
  let tni* := minimum of  tni of components i
  if tni* < xeitp then
     execute the event in i*
     distribute outputs to influenced components k
     and immediatly process them
  elseif exists component j with tnj < eitj then
     execute the event in j
     distribute outputs to influenced components k
     if this external event in k is safe
        then immediatly process it
        else schedule external event in k
  else
     optimistically execute the event in i*
     distribute outputs to influenced components k
     and immediatly process them
  endif
endloop
```

2.3 Global Control Mechanism

The global control mechanism, i.e., synchronization and event processing between different processors, can be divided into two tasks: (1) to distribute the output time estimates to other processes, (2) to distribute the output values to other processes in case of interprocessor couplings.

The first task is similar to null-message passing in other conservative synchronization mechanisms. It is a critical point of the scheme. On one side null-message traffic has to be kept low, on the other side the knowledge of the input time estimates is crucial for progress in local event processing. Several different strategies are possible and will be tested. Output time estimates might be communicated as soon as new values are available resulting in very accurate estimates but message overhead might be high. Output time estimates might be communicated only with the real output values. Here message overhead is low but estimates are not accurate always. Between these two extremes several variations are possible and to find a good tradeoff will be the subject of performance analysis.

Real data will be transmitted to the receiver as soon as they are safe, i.e., output values produced by events processed optimistically are not distributed. This has the advantage that rollbacks are kept local to the process and therefore are not too expensive. The inputs which come from other processes can be processed in the same way as all other inputs. If the resulting external event is safe to process, it can be processed immediately, if it is not safe, the external event is scheduled to occur at the event time. However, inputs from other processes might result in a rollback. Before the event is executed, the state of the component is reset to the value prior to the new external event. Output events which have been produced by this component since that time also have to be rolled back. This can be accomplished by telling the receiving components to roll back their states to a time prior to the outputs (which again might propagate the rollback to their influenced components). In this rollback process outputs to other processes, which are queued waiting for distribution because they are unsafe, might be deleted.

2.4 Numerical Integration

The numerical integration of continuous states of components assigned to different processors can be done in parallel but each computation requires synchronization and exchange of the numerical data in the case a continuous coupling between the two components exists. However, if no continuous but only discrete couplings exist between different model parts, these model parts can be integrated independently and only have to be synchronized at event times. In the same way as the sequential simulation scheme [18] identifies different numerical clusters where numerical clusters are only coupled through events, the parallel simulation scheme employs numerical clusters. Numerical clusters can be integrated in parallel only communicating and synchronizing at event times. In the current work we focus on this type of parallelism in numerical integration.

Numerical integration of continuous behavior of combined discrete/continuous DEVS-based models fits nicely into the event processing scheme described above. It can be done in a similar way as local event processing. The continuous states are integrated from one event occurrence to the next. In that, numerical integration might be safe if the earliest input time is greater than the integration time. But integration also continues if it is not safe to do so. That means that continuous states might have to be rolled back in the case a straggler input event from another process occurs. However, optimistic integration should have a high potential in several applications (as in Sparse Output DEVS [12]).

3 Relation to Other Distributed Simulation Protocols

The simulation protocol introduced above came into being by gathering many ideas emerged in the past 15 years of distributed simulation research. In their seminal work, Chandy and Misra introduced basic concepts for conservative distributed simulation [1, 2]. They defined causality requirements for correct distributed execution of events and schemes to fulfill those. Deadlocks which can occur are either recovered by a deadlock recovery scheme or avoided by special synchronization messages - so-called *null-messages.* Our conservative mechanisms is based on those ideas. In particular, conservatively executed events fulfill the causality requirement and computation of input time estimates is similar to null-message based deadlock avoidance.

Several variations of the Chandy-Misra approach have been developed. These new schemes try to exploit lookahead capabilities of the models extensively. Notable impact to our research has had the Bounded Lag Algorithm of Lubachevsky [13], the global window synchronization scheme of Nicol [14, 15], the Yaddes algorithm [5], and the shared memory implementation of Wagner, Lazowska, and Bershad [22]. Lubachevsky's algorithm is synchronous which, with every synchronization cycle, computes input time estimates for each simulator process based on *minimum propagation delays* - static lookahead assumptions - and *opaque periods* - dynamic lookahead. To limit the overhead needed to compute the input time estimates, the *bounded lag restriction* bounds the difference in the local simulation times of all simulator processes from above by a known finite constant B.

The global window synchronization algorithm [14, 15] is another synchronous algorithm which tries to exploit static and dynamic lookahead capabilities to define *global time windows* during which event processing is safe. Similar to our scheme the algorithms is tailored for low grained parallelism.

The Yaddes algorithm [5] uses a dataflow network to compute input time estimates to guarantee safe event processing. The algorithm in particular is tailored for network models which have a lot of feedback loops, like digital logic models, and therefore are difficult to parallelize using Chandy-Misra or Time Warp.

Wagner, Lazowska, and Bershad [22] improved the basic Chandy-Misra approach for implementation on a shared memory multiprocessor. Access to states of other processes through shared memory is exploited to compute better estimates of earliest input times. An *artificial blocking mechanism* is built into the run time kernel of the multiprocessor system to reduce overhead to awaken blocked processes. A centralized scheduler is used to make deadlock detection trivial and deadlock breaking inexpensive. A similar approach for parallel simulation on shared memory multiprocessors with special emphasis on exploitation of dynamic lookahead has been developed by Cota and Sargent [4].

Optimistic distributed simulation has been introduced by Jefferson [9]. Our optimistic algorithm differs from the Time Warp approach as it does not send messages to other processors optimistically. Therefore, it is a riskfree optimistic scheme related to the Breathing Time Buckets algorithm of Steinman [20] and the approach of Dickens and Reynolds [6]. Riskfree optimistic schemes do not require antimessages to annihilate incorrect messages. Rollback is kept simple. Rollback in our scheme is kept local to the process and therefore can be done in an efficient way similar to [8]. Our approach also differs from other optimistic schemes in the way fossil collection is done. Most optimistic schemes base the fossil collection on the Global Virtual Time (GVT) which is the minimum of the event times in the system. Our approach takes the earliest input time of each component as a better estimate.

Strong similarities of our approach exist to the

ADAPT system recently introduced by Jha and Bagrodia [10]. ADAPT is a simulation protocol which combines conservative and optimistic strategies. It distinguishes a *local control mechanism* and a *global control mechanism* and provides different variants of those which can be combined freely. Based on the model and workload characteristics in hand, the protocol adapts by selecting the most appropriate variants. One variant for the global control mechanism is based on computation of earliest input time estimates in the same way as in our scheme.

Our scheme also has been influenced by former efforts to parallelize simulation of DEVS models. Christensen [3] has implemented a DEVS simulator in Ada and the Time Warp operating system. Recently a new distributed simulation algorithm for networks of Sparse Output DEVS has been introduced [12]. Sparse Output DEVS are DEVS where outputs occur infrequently. Simulator processes synchronize at output times with their superior coordinator. It is riskfree optimistic with local rollback only.

4 Object Oriented Implementation

The modeling formalism and the parallel simulation concepts described above are implemented using C++ and PVM. In this section we first discuss major design decisions of the implementation of the modeling concepts, then some critical points in the realization of the parallel algorithm using PVM, and finally advantages of various hardware platforms.

4.1 DEVS-Based Modeling in C++

A major design decision in the realization of the modeling formalisms was that, in distinction to other implementations like DEVS-Scheme [23] or STIMS [18], atomic models are defined by class definitions and not by instances of formalism classes. This has the advantage that the concepts of object oriented programming, like information hiding, inheritance, and polymorphism, are ready to use for simulation modeling. In this approach hence, input and output ports and state variables are defined by member variables and state transition and output functions by member functions of atomic model classes. In the following the definition of an atomic model *Processor* is shown.

```
class Processor : public AtomicDevs {
 public:
   Input<int> in;
   Output<double> out;
   StateVar<int> reg[NREGS];
   virtual void externalTransFn (double e,
                                 InputPort &inport);
   virtual void internalTransFn (void);
   // ...
};
```

For input and output port as well as for state variables declaration generic classes - templates - are provided which are paramertized by the type of the input, output, and state variable values. These templates define access functions for reading and writing of values which also realize different tasks of the simulation protocol fully transperant to the modeler. So, the writer functions to output ports are responsible for the distribution of the output values to their destination input ports and, consequently, for the execution of the external event in the influenced component. Also, input and output port objects know methods to pack and unpack their values for interprocessor communication. The writer function of state variable objects realizes an incremental state saving and rollback mechanism.

In distinction to atomic models, modular hierarchical coupled models are not defined by class definitions but are defined employing a separate modeling language. In this language, which has a similar syntax as C++ class definitions, the modular hierarchical model structure and the couplings are defined with references to atomic model classes at the leaves of the hierarchy. Modular, hierarchical models specified in the language are transformed to C++ code. The translation process flattens the structure, instantiates only atomic components, and realizes the couplings by direct port to port connections. The following shows the definition of a coupled model *ProcBoard* from components of type *Processor*, *Bus*, and *Memory*.

```
model Processor;   // declare
model Bus;         // atomic
model Memory;      // models

model ProcBoard : DevsNetwork {
  Output<double> pout, mout;
  Memory mem;
  Bus bus;
  Processor proc;
  // define internal couplings
  IC() {
     proc.out -> bus.pin;
     bus.pout -> proc.in;
     bus.mout -> mem.in;
     mem.out -> bus.min;
  }
  // define external output couplings
  EOC() {
     proc.out -> pout;
     mem.out -> mout;
  } };
```

4.2 Parallel Simulator using PVM

The parallel simulation system is implemented using PVM (Parallel Virtual Machine) [7]. PVM is a defacto standard message passing library designed for heterogeneous environments and supports a wide range of hardware platforms.

For parallel simulation, additionally to specifying the modular hierarchical model, the simulationist is responsible for decomposing the whole model into several clusters to run on different processors. For each cluster, the parser generates an object structure with simulator objects for each atomic component, special objects responsible for communication with other processors, and one overall *coordinator* object which manages all the event handling tasks. Figure 2 shows the object structure for a 4-component model. The simulation protocol is implemented using polymorphism in such a way that normal atomic components and the special interprocessor communication objects obey the same protocol. Hence, in event handling the different objects can be handled in equal way. Each object structure is then assigned to one processor and makes one simulator executable.

PVM is used to spawn the simulator executables on different machines and for interprocessor communication. Data, i.e. outputs of atomic models and time estimates, is sent by PVM messages. Objects have to provide virtual member functions *pack* and *unpack*. The function *pack* stores the object into a PVM message and *unpack* rebuilds the objects from the PVM message. The *pack* and *unpack* methods are provided for all standard types. For user-defined types, the *pack* and *unpack* methods have to be implemented based on these basic methods.

One great advantage of the state based approach of system based modeling and the object oriented implementation is that incremental state saving is realized easily. It is implemented fully transperant to the user in the writer function of the state variable object. Whenever the writer function is called in a transition function executed optimistically, the writer function saves the old state. In that way, only state variables modified are saved.

4.3 Target Hardware Platforms

The system currently is developed on a cluster of SUN Workstations. This will also be one of our favorite hardware platforms since workstation clusters are wildly available. Workstation clusters satisfy our objective that the parallel simulator should be used as a simulator supporting our integrated modeling and simulation environment.

To show the potential of the event handling scheme, performance studies on a Sequent Symmetry S81 shared memory computer is planned. Shared-memory multiprocessor system is a promising architecture for our simulation scheme. On shared memory architectures, interprocessor communication is fast and latency is low. This allows the different processors to synchronize frequently.

Another favorite hardware platform for the combined parallel simulation algorithm is a Convex Meta-Series supercomputer. It combines a high-end workstation cluster with a vector number cruncher. The workstation cluster will be used to do the event processing part while the numerical integration will be done on the vector processor. Through this combination we hope to dramatically speed up numerical calculation of the continuous part of the simulation.

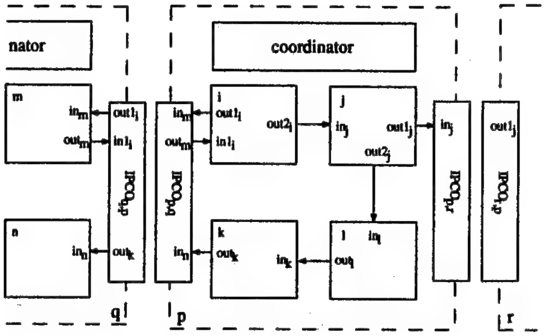


Figure 2: Object structure for process p.

5 Summary, Discussion, and Outlook

We have presented a new distributed simulation strategy for DEVS-based modular, hierarchical models. The scheme stands out as it combines conservative and riskfree optimistic strategies and is able to optimally exploit lookahead capablities of the model. Object oriented programming is employed to ease simulation modeling and make distributed simulation transparent to the user. The simulation protocol currently is under development.

Although the simulation protocol is well defined and the correctness of the conservative strategy has formally been proved [17], there are still several alternatives which will effect the performance of the simulator. One open question is, when and how often the input event estimates should be communicated to other processors, another, how far in time an optimistically executing processor should advance. Should it simulate without paying attention to the other processes until a rollback occurs or should processors synchronize. This and further questions will be answered by performance studies. First results are expected for the beginning of 1995.

References

[1] K.M. Chandy and J. Misra. Distributed simulation: A case study in design and verification of distributed systems. *IEEE Trans on Software Eng.*, 5:440–452, 1979.

[2] K.M. Chandy and J. Misra. Asynchroneous distributed simulation via a sequence of parallel

computations. *Comm. ACM*, 11:198–205, Nov 1981.

[3] E. R. Christensen and B. P. Zeigler. Hierarchical, distributed, object oriented and knowledge based simulation. In *8th Military Operations Research Society Symposium*, Annapolis, MD, 1990. US Naval Academy.

[4] B. A. Cota and R. G. Sargent. Automatic lookahed computation for conservative distributed simulation. Technical report, CASE Center, Syracuse University, Syracuse, NY, 1989.

[5] E. DeBenedictus, S Ghosh, and M.-L-Yu. A novel algorithm for discrete event simulation. *IEEE Computer*, pages 21–33, June 1991.

[6] P. Dickens and P. Reynolds. A performance model for parallel simulation. In *Proc of the 1991 Winter Simulation Conference*, pages 618–626, San Diego, 1991. SCS.

[7] A. Geist et al. PVM 3 user's guide and reference manual. Technical Report ORNL/TM/12187, Oak Ridge National Laboratory, Oak Ridge, Tennessee 37831, May 1993.

[8] R. M. Fujimoto. Time warp on a shared memory multiprocessor. *Transaction of the Society of Computer Simulation*, 6(3):211–239, 1990.

[9] D. Jefferson and H. Sowizral. Fast concurrent simulation using the time warp mechanism. In *Proc of the SCS Distributed Simulation Conf.*, pages 63–69, San Diego, 1985. Society of Computer Simulation.

[10] V. Jha and R. L. Bagrodia. A unified framework for conservative and optimistic distributed simulation. In *Proc of the 8th Workshop on Parallel and Distributed Simulation*, pages 12–19, Edinburgh, 1994. SCS.

[11] T. G. Kim and S. Park. The DEVS formalism: Hierarchical modular system specification in C++. In *Proc. 1992 European Simulation Mulitconf.*, pages 152–156, York, UK, 1992. SCS.

[12] C. Liao, A. Motaabbed, D. Kim, and B.P. Zeigler. Distributed simulation algorithms for sparce output devs. In *Proc. of AI, Simulation and Planning in High-Autonomy Systems*, Tucson AZ, Sept 1993. IEEE/CS Press.

[13] B. D. Lubachevsky. Efficient distributed event-driven simulations of multiple loop networks. *Comm of the ACM*, 32(1):111–131, 1989.

[14] D. Nicol. Performance bounds on paralel self-initiating discrete-event simulations. *ACM Trans. on Modelling and Computer Simulation*, 1(1):24–50, 1991.

[15] D. Nicol and S. Roy. Parallel simulation of timed petri nets. In *Proc. of the 1991 Winter Simulation Conference*, pages 574–583, San Diego, Dec 1991. SCS.

[16] H. Praehofer. *System Theoretic Foundations for Combined Discrete-Continuous System Simulation.* PhD thesis, Johannes Kepler University of Linz, Linz, Austria, 1991.

[17] H. Praehofer. Distributed discrete event simulation. Technical Report TR-93-1, Dept of Systems Theory, Johanens Kepler University, Linz, Austria, Jan 1993.

[18] H. Praehofer, F. Auernig, and G. Reisinger. An environment for DEVS-based multiformalims simulation in Common Lisp / CLOS. *Discrete Event Dynamic Systems: Theory and Application*, 3(2):119–149, 1993.

[19] H. Praehofer and D. Pree. Visual modeling of DEVS-based multiformalism systems based on Higraphs. In *Proc. 1993 Winter Simulation Conf.*, pages 595–603, Los Angeles, CA, Dec 1993. SCS.

[20] J. Steinman. SPEEDES: a unified framework to parallel simulation. In *Proc of the 6th Workshop on Parallel and Distributed Simulation*, pages 75–83, 1992.

[21] C. Thomas. Hierarchical object nets - a methodology for graphical modeling of discrete event systems. In *Proc. 1993 Winter Simulation Conf.*, pages 650–656, Los Angeles, CA, Dec 1993.

[22] D. Wagner, E. Lazowska, and B. Bershad. Techniques for efficient shared-memory parallel simulation. In *Distributed Simulation 1989*, pages 29–37. SCS Press, 1989.

[23] B. P. Zeigler. *Object-Oriented Simulation with Hierarchical, Modular Models.* Academic Press, London, 1990.

Abstract Simulator for the Parallel DEVS Formalism

Alex ChungHen Chow

Object Technology Products

IBM Corp.
Austin, TX 78758
alexc@austin.ibm.com

Bernard P. Zeigler
Doo Hwan Kim

Department of
Electrical and Computer Engineering
The University of Arizona
Tucson, AZ 85721
zeigler@ece.arizona.edu
dhkim@ece.arizona.edu

Abstract

A recent paper introduced the Parallel DEVS formalism which exploits the parallelism of transition collisions in the simulation of DEVS models. Here we present a design for the abstract simulator needed to prove the formalism's soundness and to serve as a reference for implementation. The abstract simulator is composed of cooperating simulation engines, (simulators and co-ordinators) that use bag-like messages to synchronize the parallel activities that are distributed across autonomous asynchronous processors. The approach suggests engines that are efficient in both sequential and distributed/parallel environments. After describing the abstract simulator we briefly discuss a prototype implementation that affords a high degree of flexibility by mechanizing the "closure under coupling" property of the Parallel DEVS formalism and the characteristics of object-oriented systems.

Keywords:
Discrete Event Simulation, DEVS formalism, Object-Oriented modeling and simulation, Distributed/parallel simulation.

1 Introduction

Hierarchical modeling capability is increasingly being recognized. The advantages of hierarchical modeling capability such as reduction in model development time, support for reuse of a database of models, and aid in model verification and validation are becoming well accepted[10]. Environments supporting hierarchical modeling are transitioning from research [16][7][9][4] into practice[6][3].

The necessary compute power for executing complex hierarchical models lies in distributed and parallel simulation[2][8][5]. Thus it is timely to reexamine the basic formalisms of discrete event modeling in the light of future high performance simulation requirements.

The *Discrete Event System Specification*(*DEVS*) formalism was introduced in the early 70's and later extended to enable constructing discrete event simulation models in a hierarchical, modular manner[14][15]. *DEVS* introduces a strong modularity between model specification and simulation. Not only does it provide a powerful modeling methodology but also a framework for model behavior generation via its abstract simulator concepts[16]. Since it is language and platform independent, *DEVS* affords an excellent vehicle for investigating alternative parallel/distributed mappings and architectures[17][13] [12].

Parallel DEVS(*P-DEVS*)[1] is a revision of the hierarchical, modular *DEVS* modeling formalism. The revision distinguishes between transition collisions and ordinary external events in the external transition function of *DEVS* models. Such separation enables us to extend the modeling capability of the collisions. The revision also does away with the necessity for tie-breaking of simultaneously scheduled events, as embodied in the *select* function (a heritage of the sequential simulation paradigm in which *P-DEVS* originated). The latter is replaced by a well-defined and consistent formal construct that allows all transitions to be simultaneously activated. The revision provides a modeler with both conceptual and parallel-execution benefits.

An earlier article[1] presented the *P-DEVS* formalism and showed it to be closed under coupling, thus preserving hierarchical, modular, construction prop-

0-8186-6440-1/94 $04.00 © 1994 IEEE

erties. This construct leads to the definition of its abstract simulator which correctly implements the formalism and exploits the increased parallelism. Here, we briefly review the *P-DEVS* formalism and proceed to discuss the abstract simulator concepts that form the basis of its concrete implementation.

2 The parallel DEVS

The *P-DEVS* model is a structure:

$$M =< X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta, >$$

X: a set of input events.
S: a set of sequential states.
Y: a set of output events.
$\delta_{int} : S \rightarrow S$: internal transition function.
$\delta_{ext} : Q \times X^b \rightarrow S$: external transition function,
  X^b is a set of bags over elements in X,
  $\delta_{ext}(s, e, \phi) = (s, e)$.
$\delta_{con} : S \times X^b \rightarrow S$: confluent transition function.
$\lambda : S \rightarrow Y^b$: output function.
$ta : S \rightarrow R_{0+\rightarrow\infty}$: time advance function,
  where $Q = \{(s, e) | s \in S, 0 < e < ta(s)\}$,
    e is the elapsed time since last state transition.

The *P-DEVS* formalism enables a modeler to explicitly define the collision behavior by using the so-called confluent transition function, δ_{con}. δ_{con} gives the modeler complete control over the collision behavior when a component receives events at the time of its internal transition, $e = 0$ or $e = ta(s)$. Rather than serializing model behavior at collision times, the *P-DEVS* formalism leaves this decision of what serialization to use, if any, to the modeler. Indeed, if so desired, the *E-DEVS*[11] formalism can be recovered by setting $\delta_{con}(s, x^b)$ to $\delta_{ext}(s_n, 0, x_n)$, where $n \geq 1$, $s_1 = \delta_{int}(s)$, $s_n = \delta_{ext}(s_{n-1}, 0, x_{n-1})$ when $n > 1$, and x_n is a desired serialization defined by $Order(x^b)$.

The semantics of the *Parallel DEVS* are as follows: the internal transitions are carried out at the next event time for all imminent components receiving no external events. Also, external events generated by these imminents trigger external transitions at receptive non-imminents (those components for which there are no internal transitions scheduled at the event receiving time). However, for those components for which the internal and external transitions collide, the *confluent transition function* is employed instead of either the internal or external transition function to determine the new state.

The structure of the revised *coupled model* is —

$$DN =< X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} >$$

X: a set of input events.
Y: a set of output events.
D: a set of components.
for each i in D,
  M_i is a component.
for each i in $D \cup \{self\}$, I_i is the influencees of i.
for each j in I_i,
  $Z_{i,j}$ is a function,
  the i-to-j output translation.

The structure is subject to the constraints that for each i in D,
$M_i =< X_i, S_i, Y_i, \delta_{inti}, \delta_{exti}, \delta_{coni}, ta_i >$ is a *P-DEVS* structure,
I_i is a subset of $D \cup \{self\}$, i is not in I_i,
$Z_{self,j} : X_{self} \rightarrow X_j$,
$Z_{i,self} : Y_i \rightarrow Y_{self}$,
$Z_{i,j} : Y_i \rightarrow X_j$.

Here $self$ refers to the coupled model itself and is a device for allowing specification of external input and external output couplings.

Closure of the *P-DEVS* formalism under coupling was done by constructing the resultant of a coupled model and showing it to be a well defined *P-DEVS*. The *resultant* of a coupled model ($DN =< X, Y, D, \{M_i\}, \{I_i\}, \{Z_{i,j}\} >$) is a *P-DEVS* model ($M =< X, S, Y, \delta_{int}, \delta_{ext}, \delta_{con}, \lambda, ta >$), where
$S = \times Q_i$ where $i \in D$.
$ta(s) = \text{minimum}\{\sigma_i | i \in D\}$,
  where $s \in S$ and $\sigma_i = ta(s_i) - e_i$.
Let
  $s = (..., (s_i, e_i), ...)$,
  $IMM(s) = \{i | \sigma_i = ta(s)\}$,
  $INF(s) = \{j | j \in \cup_{i \in IMM(s)} I_i\}$,
  $CONF(s) = IMM(s) \cap INF(s)$,
  $INT(s) = IMM(s) - INF(s)$,
  $EXT(s) = INF(s) - IMM(s)$.

We partition the components into four sets at any transition time. $INT(s)$ contains the components ready to make an internal transition without input events. $EXT(s)$ contains the components receiving input events but not scheduled for an internal transition. $CONF(s)$ contains the components receiving input events and also scheduled for internal transitions at the same time. $UN(s)$ contains the remaining components. Then,
$\lambda(s) = \{Z_{i,self}(\lambda_i(s_i)) | i \in IMM(s) \wedge self \in I_i\}$.

$\delta_{int}(s) = (\ldots, (s'_i, e'_i), \ldots)$,
where
$(s'_i, e'_i) = (\delta_{int\,i}(s_i), 0)$ for $i \in INT(s)$,
$(s'_i, e'_i) = (\delta_{ext\,i}(s_i, e_i + ta(s), x^b_i), 0)$ for $i \in EXT(s)$,
$(s'_i, e'_i) = (\delta_{con\,i}(s_i, x^b_i), 0)$ for $i \in CONF(s)$,
$(s'_i, e'_i) = (s_i, e_i + ta(s))$ otherwise $i \in UN(s)$,
and
$x^b_i = \{Z_{o,i}(\lambda_o(s_o)) | o \in IMM(s) \wedge i \in I_o\}$.

The resultant internal transition comprises of four kinds of component transitions: internal transitions of $INT(s)$ components, external transitions of $EXT(s)$ components, confluent transitions of $CONF(s)$ components and the remainder,$UN(s)$, whose elapsed times are merely updated by $ta(s)$.

The δ_{ext} of the resultant is defined by:
$\delta_{ext}(s, e, x^b) = (\ldots, (s'_i, e'_i), \ldots)$,
where
$(s'_i, e'_i) = (\delta_{ext\,i}(s_i, e_i + e, x^b_i), 0)$ for $i \in I_{self}$,
$(s'_i, e'_i) = (s_i, e_i + e)$ otherwise,
and
$x^b_i = \{Z_{self,i}(x) | x \in x^b \wedge i \in I_{self}\}$.

The incoming event bag, x^b is translated and routed to the event bag, x^b_j, of each influenced child, j. The resultant's external transition comprises all the external transitions of the influenced children.

Finally, the δ_{con} of the resultant is defined by:
Let
$INF'(s) = \{j | j \in \cup_{i \in (IMM(s) \cup \{self\})} I_i\}$,
$CONF'(s) = IMM(s) \cap INF'(s)$,
$INT'(s) = IMM(s) - INF'(s)$,
$EXT'(s) = INF'(s) - IMM(s)$.
$\delta_{con}(s, x^b) = (\ldots, (s'_i, e'_i), \ldots)$,
where
$(s'_i, e'_i) = (\delta_{int\,i}(s_i), 0)$ for $i \in INT'(s)$,
$(s'_i, e'_i) = (\delta_{ext\,i}(s_i, e_i + ta(s), x^b_i), 0)$
for $i \in EXT'(s)$,
$(s'_i, e'_i) = (\delta_{con\,i}(s_i, x^b_i), 0)$ for $i \in CONF'(s)$,
$(s'_i, e'_i) = (s_i, e_i + ta(s))$ otherwise,
and
$x^b_i = \{Z_{o,i}(\lambda_o(s_o)) | o \in IMM(s) \wedge i \in I_o\} \uplus$
$\{Z_{self,i}(x) | x \in x^b \wedge i \in I_{self}\}$.

The critical difference in the *P-DEVS* compared with the original DEVS is that to establish closure under coupling, we must also define the δ_{con} of the resultant. Fortunately, it turns out that the difference between δ_{con} of the resultant and its δ_{int} is simply the extra confluent effect produced by the incoming event bag, x^b, at simulation time $ta(s)$. By redefining the influencee set to $INF'(s)$ that includes the additional influencees from the incoming couplings, $z(self, i)$, we come up with three similar groups for δ_{con}. The hierarchical consistency is achieved here by the $\uplus$ operation that gathers all external events, whether internally or externally generated, at the same time into one single event group.

¿From the definition of the δ_{int}, δ_{con}, and δ_{ext}, we see that they are special cases of a more generic transition function $\delta(s, e, x^b)$[15]. δ_{int} is applied to the cases when $(s, e, x^b) = (s, ta(s), \phi)$, δ_{con} to the cases when $(s, e, x^b) = (s, ta(s), x^b)$ where $x^b \neq \phi$, and, δ_{ext} to (s, e, x^b) where $0 \leq e < ta(s)$ and $x^b \neq \phi$.

3 The abstract simulator

We now describe the abstract simulator needed to demonstrate soundness of the *P-DEVS* formalism. As in the original definition, we specialize the processors into two different simulation engines, *simulator* and *co-ordinator* [15].

Both δ_{con} and δ_{ext} depends on the events in the bag, x^b. An event in the bag is a result from an output function and all the translations on the event path. An output function depends on a state prior to a transition at the same instance. It is clear that the output function must be invoked before any transition function. We use $(@, t)$ and $(done, t)$ messages to synchronize this activity while (y, t) and (q, t) messages trasport the output content. We also assume that if two messages are sent from the same source, the ordering between them is preserved at the receiving end.

The *simulator* attached to an atomic model is given first:

```
when a $(@, t)$ message is received
if $t = t_N$ then
  $y := \lambda(s)$
  send $(y, t)$ to the parent coordinator
  send $(done, t)$ to the parent coordinator
end if
else raise error
end when

when a $(q, t)$ message is received
lock the bag
Add event $q$ to the bag
unlock the bag
send $(done, t)$ to the parent coordinator
end when

when a $(*, t)$ message is received
```

```
case $t_L \le t < t_N$ and $bag$ is not empty
  $e := t - t_L$
  $s := \delta_{ext}(s, e, bag)$
  empty $bag$
  $t_L := t$
  $t_N := t_L + ta(s)$
end case
case $t = t_N$ and $bag$ is empty
  $s := \delta_{int}(s)$
  $t_L := t$
  $t_N := t_L + ta(s)$
end case
case $t = t_N$ and $bag$ is not empty
  $s := \delta_{con}(s, bag)$
  empty $bag$
  $t_L := t$
  $t_N := t_L + ta(s)$
end case
case $t > t_N$ or $t < t_L$
  raise error
end case
send $(done, t_N)$ to parent coordinator
end when
```

The *simulator* uses one single message, $(*, t)$, to synchronize three different transitions of the atomic model. Obviously, other implementations with more synchronization messages for different transitions can remove the need of case statements for possibly faster simulation. The implementation introduced here serves as an example to indicate the correct semantic application of each transition function which enables us to use the generic transition function described above for a *co-ordinator*. The implementation of a *co-ordinator* is given.

```
when a $(@, t)$ message is received from parent coordinator
if $t = t_N$ then
  $t_L := t$
  for all imminent child processors $i$ with minimum $t_N$
  send $(@, t)$ to child $i$
  cache $i$ in the $synchronize$ set
  end for
  wait until $(done, t)$'s are received from all imminent processors
  send $(done, t)$ to the parent coordinator
else raise an error
end when
```

```
when a $(y, t)$ message is received from child $i$
for all influencees, $j$ of child $i$
  $q := z_{i,j}(y)$
  send $(q, t)$ to child $j$
  cache $j$ in the $synchronize$ set
end for
wait until all $(done, t)$'s are received from $j$'s
if $self \in I_i$ (y is to be transmitted upward) then
  $y := z_{i,self}(y)$
  send $(y, t)$ to the parent coordinator
end if
end when
```

```
when a $(q, t)$ message is received from parent coordinator
lock the $bag$
Add event $q$ to the $bag$
unlock the $bag$
end when
```

(y, t) messages are always processed within the *wait* statement when receiving a $(@, t)$ message. This synchronization ensures that the outputs of any model, either atomic or coupled, are routed to their immediate influencees' bags. All children ready for a transition are cached in a set called *synchronize* set to eliminate the activities of $UN(s)$ components. The elapsed time can always be calculated from the t_L associated with each component and the absolute global clock, t.

From the construction described in the previous section, we see that
$x_i^b = \{z_{o,i}(\lambda_o(s_o)) | o \in IMM(s) \wedge z_{o,i} \in Z\} \uplus \{z_{self,i}(x) | x \in x^b\}$

After the processing of $(@, t)$ is over, output events are distributedly stored in input bags of influencees throughout the hierarchy. Though the first part of x_i^b is ready now, the $\uplus$ with the second part must be done by sending (q, t) messages to influencees of $self$ at the beginning of the each $(*, t)$ phase. This operation assures the uniformity of the hierarchy. All events are routed down to the atomic influencees by successive $(*, t)$ phases of nodes from root to atomic components. A transition is completed when finally one of the transition functions is invoked at the atomic level.

```
when a $(*, t)$ message is received from parent coordinator
if $t_L \le t \le t_N$ then
  for all receivers, $j \in I_{self}$ and all $q \in bag$
    $q := z_{self,j}(q)$
    send $(q, t)$ to $j$
    cache $j$ in the *synchronize* set
  **end for**
  empty *bag*
  wait until all $(done, t)$'s are received
  **for** all $i$ in the *synchronize* set
    send $(*, t)$ to $i$
  **end for**
  wait until all $(done, t_N)$'s are received
  $t_L := t$
  $t_N :=$ minimum of components' $t_N$'s
  clear the *synchronize* set
  send $(done, t)$ to parent coordinator
else raise an error
end when
```

Elements in the *synchronize* set are imminent components, influencees or both. Because of the consistent application, we delay the distinction of a transition only until the notification arrives at the atomic level.

The implementation of this coordinator routes down the output events during the $(*, t)$ phases. It simply reflects the construction of the transition functions of a coupled model. Another implementation might choose to route the events during the $(@, t)$ phase directly to the final atomic influencees. The bag implementation of the coupled model can thus be omitted. Both implementations are equivalent and render the same simulation result.

The topmost coordinator is driven by a special coordinator called the root coordinator which constantly advances the global simulation time to the next simulation time of a simulation, sends $(@, t)$ and $(*, t)$ messages to the topmost coordinator, asks the next simulation time, and repeats until the next simulation time is infinite.

```
Root coordinator
$t := t_N$ of the topmost coordinator
while $t \neq \infty$
  send $(@, t)$ to the topmost coordinator
  wait until $(done, t)$ is received from it
  send $(*, t)$ to the topmost coordinator
  wait until $(done, t_N)$ is received from it
end while
raise simulation completed
```

The simulation procedure exposes the parallelism among transitions of elements in *synchronize* set and abstract simulator design handles *transitory* states in a well defined manner.

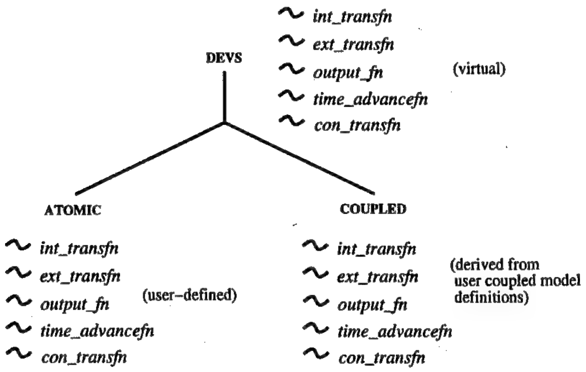


Figure 1: Abstract Class and Inheritance Hierarchy Exploiting Closure Under Coupling

4 Flexibility of hierarchical model mappings

The standard mapping of a hierarchical *P-DEVS* model onto an abstract simulator results in a hierarchical architecture with a one-one correspondence to the models composition tree structure. However, many alternative mappings exist and some are more likely much better depending on the model behavior and the receiving platform characteristics. Some possibilities have been investigated[17][13]. Here, we note that closure under coupling enables any coupled model in the model composition tree to be mapped into an equivalent resultant model. This mapping, described above, can be implemented within an object-oriented framework as illustrated in Figure 1.

Here, atomic model and coupled model objects present the same interface to clients, one that is abstracted in a devs superclass with virtual transition functions. This greatly increases the flexibility with with mappings can be done. In partcular it helps overcome limitions of conventional high performance architectures which do not support hierarchical clusters. Only one coordinator process is needed at the top level for managing the intercommunication and synchronization of nodes. The same simulator processes run on other nodes and are linked to either atomic or coupled models, the resultant mapping embodied in the common interface blinds them to the difference.

We are currently investigating the application of this concept to large scale ecosystem simulation. As illustrated in Figure 2, a landscape, such as a watershed, is represented by a "base" model of a large number,e.g., one million, of cells. This number is orders of magnitude larger than the number of nodes in the highest performance massively parallel computers such the 1000 node CM-5. Therefore the base model cannot be mapped in a one-one manner and some partitioning is required as illustrated in Figure 2. To retain the base model dynamcs, each block becomes a coupled model over the component cells within its scope. Using the closure under coupling concept, each coupled model is represented by its resultant *P-DEVS* model and these resultants are coupled together in a manner preserving the coupling behavior of orginal base model. The outputs of blocks are collections of outputs of their enclosed cells and their management is nicely handled by the bag construct in the *P-DEVS* formalism. The block resultants are assigned to simulators and the coupling of these resultants to a coordinator. The transformation of the base model into an equivalent coupling of blocks is called "deepening" and the entire process can be defined formally and implemented nicely in the object-oriented paradigm. Partitioning of cells into blocks is not constrained and indeed, can be performed dynamically during execution to balance the processor loads as the locus of cellular activity migrates about.

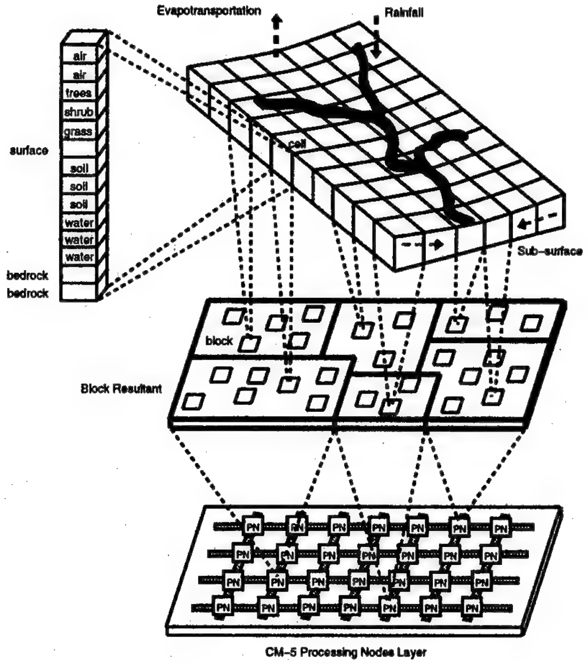


Figure 2: Mapping of landscape base model onto a flat massively parallel architecture (e.g. CM-5)

5 Conclusions

In the *Parallel DEVS* formalism, a modeler is explicitly enabled to supply the confluent transition function that captures the collision behavior. This function allows the coupling construction to follow the semantics of a collision down to the atomic level and obviates any behavioral difference between a model and its deepened and flattened restructurings.

The abstract simulator concept leads to many possible implementations. The well isolated transition groups add to the existing possibilities to exploit the parallelism of the hierarchical *DEVS* models. Since the abstract simulation engine is based on the assumption of a parallel environment, the implementation on parallel machines is straightforward. Moreover, closure under coupling supports flexible restructuring for more effective mappings, both static and dynamic, to particular platforms.

Acknowledgements

Some of this research is supported by NSF HPCC Grand Challenge Application Group Grant ASC-9318169 with ARPA participation and employs the CM-5 at NCSA under grant MCA94P02.

References

[1] Alex C. Chow and Bernard P. Zeigler. Parallel DEVS: A parallel, hierarchical, modular modeling formalism. In *Winter Simulation Conference Proceedings*, Orlando, Florida, 1994. SCS.

[2] Richard M. Fujimoto. Parallel discrete event simulation. *Communications of the ACM*, 33(10):30–53, 1990.

[3] J. Hu. Cedes: Object-oriented hardware modeling & simulation.

[4] Tag Gon Kim. *DEVSIM++ User's Manual.* Taejon, Korea, 1994.

[5] B. Lubachevesky, A. Weiss, and A. Schwartz. An analysis of rollback-based simulation. *ACM Transactions on Modeling and Computer Simulation*, 1(2), 1991.

[6] C. D. Pegden and D. A. Davis. Arena^{TM} : A SIMAN/CINEMA based hierarchical modeling system. In *Winter Simulation Conference Proceedings*, Phoenix, AZ, 1992.

[7] H. Praehofer. An environment for DEVS-based multiformalism simulation in common Lisp/CLOS. *Discrete Event Dynamic Systems*, 3, 1993.

[8] B. Preiss. The Yaddes distributed discrete event simulation specification language and execution environment. In *Distributed Simulation 89*. SCS Press, 1989.

[9] J. W. Rozenblit and P. Janknski. An integrated framework for knowledge-based modeling and simulation of natural systems. *Simulation*, 57(3), 1990.

[10] Robert Sargent. Hierarchical modeling for discrete event simulation (panel). In *Winter Simulation Conference Proceedings*, page 569, Los Angeles, CA, 1993.

[11] Yung-Hsin Wang and Bernard P. Zeigler. Extending the DEVS Formalism for Massively Parallel Simulation. *Discrete Event Dynamic Systems: Theory and Applications*, 3:193–218, 1993.

[12] T.G. Kim Y.R. Seong and K.H. Park. Mapping modular, hierarchical discrete event models in a hypercube multicomputer. In *Proceedings of International Conference on Massively Parallel Proc. Application and Development*, 1994.

[13] T.G. Kim Y.R. Seong, S.H. Jung and K.H. Park. Parallel simulation of hierarchical modular devs models: A modified time warp approach. *International Journal of Computer Simulation*, (to appear).

[14] Bernard P. Zeigler. *Theory of Modelling and Simulation*. Wiley-Interscience, New York, 1976.

[15] Bernard P. Zeigler. *Multifacetted Modelling and Discrete Event Simulation*. Academic Press, London, 1984.

[16] Bernard P. Zeigler. *Object-Oriented Simulation with Hierarchical, Modular Models*. Academic Press, San Diego, California, 1990.

[17] B.P. Zeigler and G. Zhang. Mapping hierarchical discrete event models to multiprocessor systems: Concepts, algorithm and simulation. *Parallel & Distributed Computing*, 10:271–281, 7 1990.

An Approach to Object-Oriented Modeling and Performance Evaluation

Lien-Pharn Chien

Department of Information Engineering
Kaohsiung Polytechnic Institute
Kaohsiung, Taiwan

Jerzy W. Rozenblit

Electrical and Computer Engineering
The University of Arizona
Tucson, AZ 85721, U.S.A.

Abstract

Within the past few decades, diverse modeling and simulation tools have been applied in extensive applications. The approaches used range from programming with a specific simulation description language to automation using an icon-driven user interface. The advantage in utilizing simulation is to assess the system's performance prior to an actual implementation. Functionality, maintainability, and expansibility are the primary criteria used to make a choice of a specific tool. To strengthen these criteria, a general-purpose environment called Performance Object-oriented modeling and Simulation Environment (POSE) has been developed. The objectives of POSE are to automatically construct simulation models for the systems to be designed, to efficiently define the system performance measures, and to accurately generate the performance data expected. The environment is briefly summarized and an application study for a multiprocessor computer system is presented.

1 Introduction

As fiber optics, ultra large-scale integrated circuits, asynchronous transfer mode, and more advanced technologies are introduced, new application systems have become much more complex. The behavior of the systems is usually of high complexity and is difficult to evaluate by *analytical approaches.* It is believed that if no analytical approaches can be applied, constructing a new, complex system can be expensive, time consuming, and risky. Therefore, *simulation* or *hybrid approaches* are explored to mitigate the problems [7]. This is the first reason triggering this work. In addition, it is necessary to construct a model required before carrying out system simulation. To build the system model in most of existing simulation languages, users (or system designers) must know the syntax of a specific language and how to program the model correctly. The situation motivates our research to devise a way that allows users to do system modeling without the knowledge of an underlying simulation language.

Furthermore, the object-oriented (for short, OO) concept has shown a great potential in extensive applications, especially the advantages of reuse and maintainability. The third characteristic in *POSE* is to utilize the OO concept in cooperation with Queueing Theory [4] and the structure of DEVS formalism [13] such that each *POSE*'s model has a concrete configuration and is efficient in processing the problems of system performance measures. Lastly, we strive to improve *POSE* to strengthen its functionality and expansibility. This is achieved through the design of the hierarchical model-base management. The hierarchical model bases are established by dividing the modeling procedure into two parts: the system-architecture and the system-performance modeling stages.

2 The Design of POSE

As described in detail in [1], the design concept focuses on automating the simulation model creation and providing the required performance calculation and evaluation. *POSE's* architecture is depicted in Figure 1. The arrows in the figure show the operation flow in *POSE.* Each item beside an asterisk expresses the basic part corresponding to the stage right above or below it. According to the flow, the requirements and constraints of the system to be developed are considered first. The requirements include performance objects (indices) like throughput, utilization and turnaround time. After analyzing the system's requirements and constraints, the AGEF (Automatic

0-8186-6440-1/94 $04.00 © 1994 IEEE

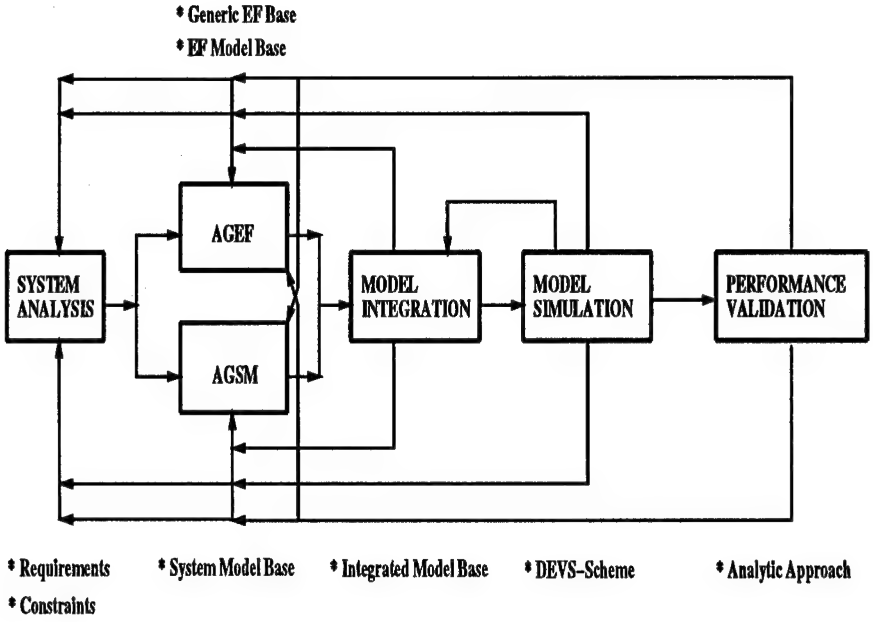


Figure 1: The Design Flow of POSE.

Generation of Experimental Frame) stage is processed in order to embed this information into an experimental frame (for short, EF) for future use [12].

At the System Analysis stage, the system's architecture is analyzed. For example, when a multiprocessor computer system is to be modelled, the information about the connections among CPUs, memory units and IO units, the characteristics of each unit, and the partitions of the system, have to be obtained after the analysis procedure. Based on this information, the AGSM (Automatic Generation of System Model) is invoked to model the computer system. As soon as the AGEF and the AGSM stages are completed, the Model Integration (for short, MI) stage takes place. A complete integrated model is then generated. This model is able to produce the performance data for the system in terms of the requirements and constraints specified in the EF(s).

The output provided by *POSE* are integrated models which are useful at the next stage, Model Simulation. All performance data are collected and computed within this stage. These data are used to validate the accuracy of the system models (e.g. the computer model) via mathematical approaches.

Three model bases, Experimental Frame Model Base (EFMB), System Model Base (SMB), and Integrated Model Base (IMB), along with a performance object-based library called Generic Experimental Frame Base (GEFB), are used to support the hierarchical modeling-automation flow. These bases have the hierarchical relationship of IMB at the root with two children SMB and EFMB. In turn, EFMB requires the resource in GEFB. They are originally empty but become populated as systems are developed in *POSE.* The power of *POSE* is enhanced by maximizing the flexibility of the execution flow feedback as referred to in Figure 1 and designing a corresponding interface shown in Figure 2 (where both *Node Modeling and System Modeling* functions comprise the AGSM stage). More details about the environment and its implementation can be found in [1]. In what follows, we focus on the model integration and simulation stage and provide an illustrative example.

3 Model Integration and Simulation

The models in the system model base (SMB) and the experimental frame model base (EFMB) can be operated in a stand alone mode with DEVS-Scheme but no meaningful output is produced. To come up with the performance metrics required for an application system, the function of Model Integration (MI) is then designed. Figure 3 shows the relationship among the SMB, EFMB and the integrated model base (IMB). This figure also points out that only integrated models are allowed to be simulated in *POSE.*

In order to carry out flexible model integration,

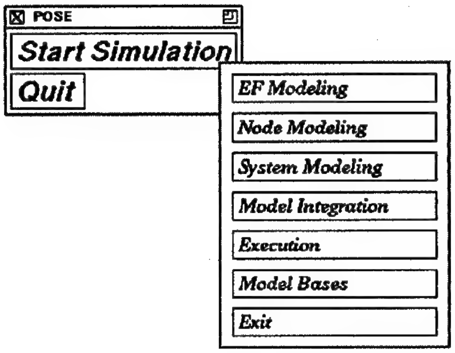


Figure 2: The Function Layout Window in POSE.

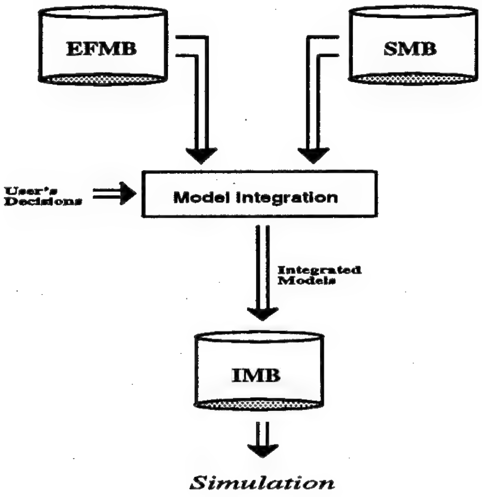


Figure 3: Model-Base Relationship in Model Integration.

the schemes of *global* and *distributed experimental frames* [8] are employed at this stage. For instance, an interconnected network like ARPANET [11] connects many Local Area Networks (LANs) via gateways. Since the role of gateways within the interconnected network is very critical and sensitive, performance measures at gateways are particularly important. In general, throughput and utilization are factors of greatest concern. This situation requires attaching different configurations of EFs to the gateways at different geographical areas. This flexibility has been carried out by using both schemes. Due to the flexible attachment of an EF to any layer or component in a system model, the EFs stored in the EFMB can be retrieved and coupled to the system model without any restriction during the processing of the MI function.

The goal of simulation in *POSE* is to generate performance data for performance evaluation. All the data expected are gathered and saved in different log files specified in the transducer(s) during simulation. Basically, there are three pre-defined log files: *job arrival*, *job finished*, and *summary*, at each transducer. Both *job arrival* and *finished* files keep the so-called raw performance data consisting of each job name and its priority with the arrival or departure time, respectively. The log files rather than the pre-defined ones are for specific purposes such as throughput, turnaround, etc. The *summary* file periodically collects all processed performance data such as:

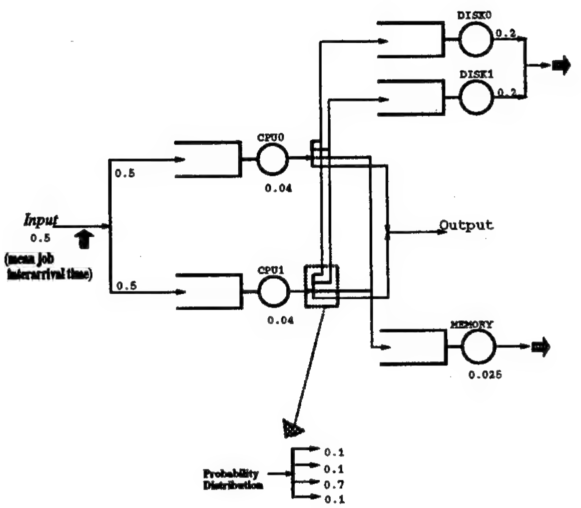
```
** ef-computer at time 320

throughput :: 2.08524590163934
turnaround :: 1.66049869504983

** ef-computer at time 340

throughput :: 2.08709677419355
turnaround :: 1.65773744063163

** ef-computer at time 360

throughput :: 2.0952380952381
turnaround :: 1.6632224979867
```

A simulation run is finished as soon as the tolerance condition pre-set by employing the technique of *terminating simulation* is met [5].

4 The Experiments in POSE

Even though *analytical approaches* provide efficient and accurate ways to process performance measures, the *simulation approach* offers an alternative when: 1) the complexity of a system prohibits deterministic results, or 2) the complexity is difficult to analyze mathematically. The *Simulation approach* also provides a means for evaluating and comparing new systems prior to their actual implementation. Nevertheless, *analytical approaches* can be used to evaluate the performance outcome generated by *POSE*.

The following simulation experiments are used to test *POSE*'s functionality and accuracy. Their simulation results are synthesized through the scheme of *confidence interval* under the control of *terminating simulation* [5]. Also, the results are evaluated by means of mathematical analysis with queueing theory.



Comment : The values beside circles are their mean job service time (ms).

Figure 4: A Multiprocessor Computer System.

A Multiprocessor Computer System: Performance evaluation in various computer systems has been studied extensively [3, 6, 7, 10]. For comparison purposes, a multiprocessor computer system is designed in *POSE*. The experiments related to the proposed computer system shown in Figure 4 are to determine how the system would perform under various changes. Based on the figure, these changes could be about the input rate (system workload), the service rates of cpus, (main) memory and disks, and the probability settings on links (buses). Since the role of the cpu and memory is more sensitive to the whole system, we are primarily concerned with changes in their rates. Due to the variety, the performance measures regarding the mean job turnaround time, i.e., average time delay, in the system are considered. These performance measures are gathered through executing the system's model, which is created by *POSE*, as shown in Figure 5.

The first experiment examines the quantity changes in turnaround time by gradually modifying the cpu's mean service time (the reciprocal of service rate). Other factors involved here have the following conditions:

1) No jobs are blocked at any receiving unit.

2) Mean job interarrival time to the system : 0.5 *millisecond/job*

3) Mean service rate of the memory : 0.025 *millisecond/job*

4) Mean service rate for each disk : 0.2 *millisecond/job*

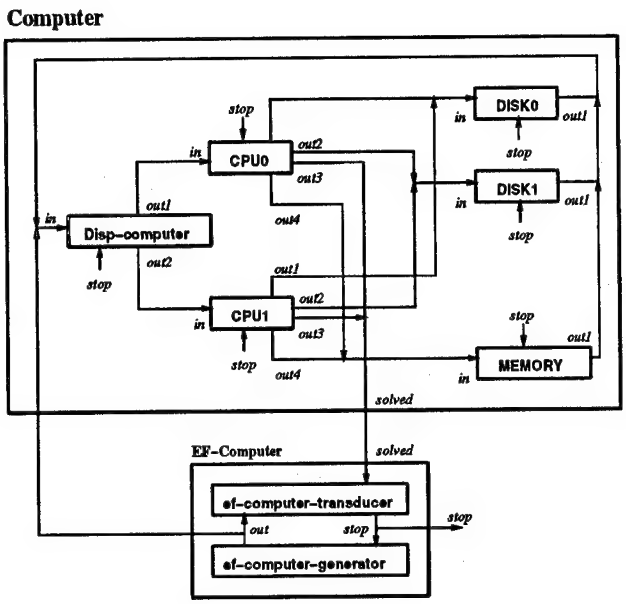


Figure 5: A Multiprocessor Computer System.

5) Input jobs are immediately dispatched to each cpu with equal probability.

6) Cpu's outgoing jobs sent to the two disks, the memory, and outside of the system with the probabilities, 0.1, 0.1, 0.7, and 0.1, respectively.

By means of multiple simulation runs at each cpu setting, Figure 6 plots the related confidence interval with 95% via two curves **Simulation-Upper** (i.e. the upper bound of the interval) and **Simulation-Lower** (i.e. the lower bound). This area between the upper and lower curves shows that *POSE* provided a good enough estimate by comparing it to results calculated with the *analytical approach.* Based on the same conditions, we proceed with the *analytical approach* by using queueing theory. Since the computer system model is an *Open Queueing Network* (OQN) with Poisson input rate, exponential service rates and infinite buffer sizes at all queues, it can be analyzed by applying Jackson's theorem [2]. The corresponding analytical curve is marked with **Analytical-0.5** for the purpose of evaluation. (The "0.5" expresses the mean job interarrival time to the system is 0.5 millisecond, i.e., the job input rate is 2 *jobs/ millisecond.*) From the curve distributions, it is concluded that:

1) The area specified by the 95% confidence interval almost covers the analytical curve except for the range close to 0.08 and up. This exceptional range results from job congestion occurred in cpus to the extent that

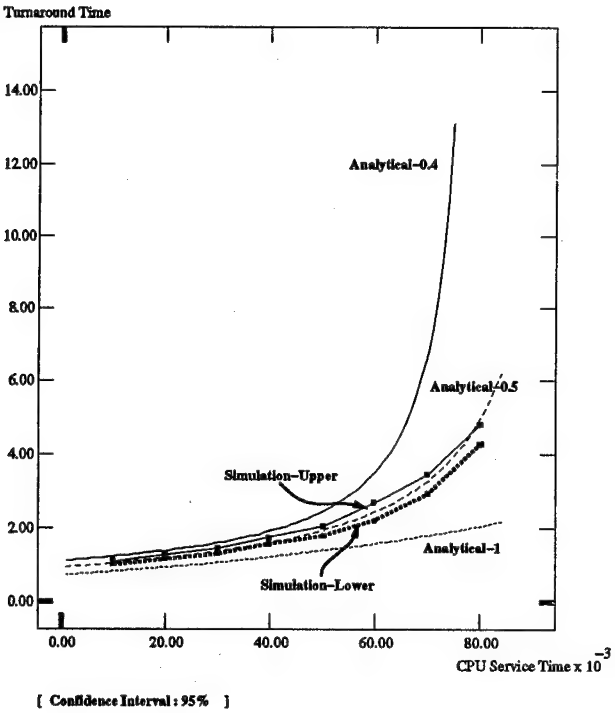


Figure 6: The Effect of The CPU's Rate Changes.

the whole system becomes unstable and the bound of 95% coverage is no longer obeyed. Therefore, this evaluation shows a high accuracy and sensitivity for the *simulation approach* performed in *POSE*. (In the figure, two other analytical curves are drawn for reference. The **Analytical-0.4** curve exhibits serious job congestion when the mean service time of a cpu is over 0.06, a situation that does not occur in the **Analytical-1** curve during the changes of cpu time. This is because a lower input rate is assigned to the latter.)

2) If the service time of a cpu is less than 0.06, then higher input rates are suggested unless there is a real-time factor.

The second experiment investigates how a change in memory service-time affects job turnaround time in the proposed system. Figure 7 illustrates the effect caused by the change. The related conditions set in the experiment are the same as in the first experiment except that : a) the cpu service-time is fixed at 0.04 *millisecond/job*, and b) the memory service-time is adjusted from 0.01 to 0.05 *millisecond/job*.

The simulation outcome is plotted by two curves **Simulation-Upper** and **Simulation-Lower** with 95% confidence interval. The related mathematical method is also built according to Jackson's theorem. The corresponding analytical outcome is drawn for comparison to the simulation outcome. Due to the nonexistence of the job congestion problem given the testing conditions, the **Analytical-0.04** curve is completely covered within the area between the two bound curves.

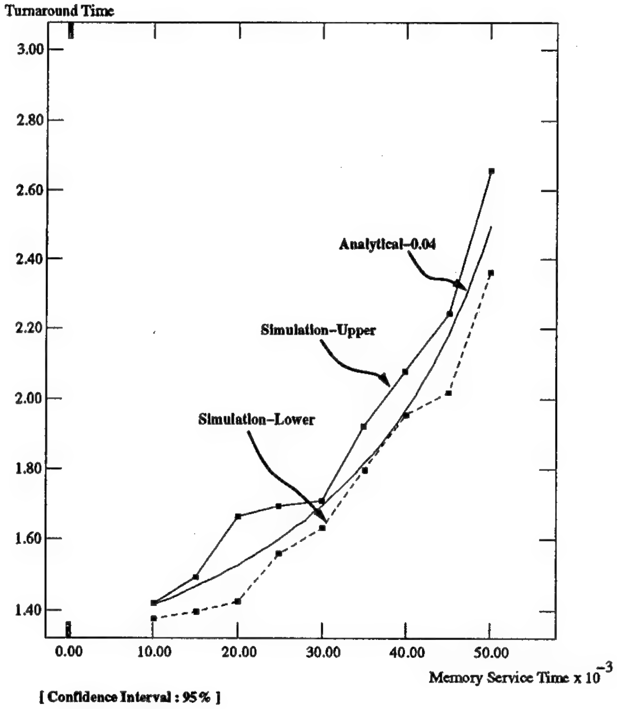


Figure 7: The Effect of The Memory's Rate Changes.

The experimental results for the proposed computer system show that performance outcomes produced by *POSE* provide accurate estimates. The same outcomes of the tests of Gordon-Newell Networks are also obtained.

5 Conclusions

In *POSE*, users can systematically construct a complex system model with a multilayer and multicomponent architecture through the interactive window-driven interface. The architecture facilitates hierarchical modeling and a hierarchical model-based management. By making connections from *POSE* to DEVS-Scheme, model simulation and performance data collection and computation are then accomplished. In conclusion, the contributions of this work to the field of modeling and simulation automation are in a) hiding of simulation language, b) modeling automation, c) hierarchical modeling, and d) effective performance measures generation.

References

[1] L.P. Chien and J.W. Rozenblit, An Environment for Automatic System Performance Evaluation, Proceedings of the 1993 Winter Simulation Conference, pp. 582-588, Los Angeles, CA December 1993.

[2] J.R. Jackson, Networks of Waiting Lines, *Operations Research*, Vol. 5, pp. 518-521, 1957.

[3] P.J. King, *Computer and Communication Systems Performance Modelling*, Prentice Hall International (UK) Ltd., 1990.

[4] L. Kleinrock, *Queueing Systems*, Vol. I, John Wiley & Sons, Inc., 1975.

[5] A.M. Law and W.D. Kelton, *Simulation Modeling and Analysis*, McGraw-Hill, Inc., 1982.

[6] M.F. Morris and P.F. Roth, *Computer Performance Evaluation: Tools and Techniques for Effective Analysis*, Van Nostrand Reinhold Company, 1982.

[7] T.G. Robertazzi, *Computer Networks and Systems: Queueing and Performance Evaluation*, Springer-Verlag New York, 1990.

[8] J.W. Rozenblit, Experimental Frame Specification Methodology for Hierarchical Simulation Modeling, *International J. of General Systems*, Vol. 19, No. 3, pp. 317-336, 1991.

[9] J.W. Rozenblit and J. Hu, Experimental Frame Generation in a Knowledge-Based System Design and Simulation Environment, *Modeling and Simulation Methodology: Knowledge Systems' Paradigms*, (eds.: M. Elzas et.al.), North Holland, pp. 451-466, 1989.

[10] B.W. Stuck and E. Arthurs, *A Computer and Communications Network Performance Analysis Primer*, Bell Telephone Laboratories, Inc., 1985.

[11] A.S. Tanenbaum, *Computer Networks*, 2nd Edition, Prentice-Hall, Inc., 1988.

[12] B.P. Zeigler, *Multifacetted Modelling and Discrete Event Simulation*, Academic Press, 1984.

[13] B.P. Zeigler, *Object-Oriented Simulation with Hierarchical, Modular Models - Intelligent Agents and Endomorphic Systems*, Academic Press, 1990.

The DEVS Formalism : A Framework for Logical Analysis and Performance Evaluation for Discrete Event Systems *

Gyung Pyo Hong and Tag Gon Kim
Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
373-1 Kusung-dong Yusung-gu, Taejon 305-701, Korea.

Abstract

This paper proposes a framework which supports performance evaluation and logical analysis of discrete event systems using a unified formalism, i.e., the DEVS(Discrete Event System Specification) formalism. For performance evaluation, DEVSim++, a realization of the DEVS formalism and the associated simulation algorithms in C++, is used. For logical analysis, the dual language approach is adopted. We use the DEVS formalism as an operational formalism to describe system's behavior. Temporal Logic(TL) is employed as an assertional formalism to specify system's properties. To reduce states space in logical analysis, we exploit a projection mechanism. The method is a mapping of a set of states in models into a state which obtained from TL assertions. An example of logical analysis for Alternating Bit Protocol is given.

1 Introduction

Systems development process consists of a formal specification of requirements, modeling from the specification, validation of the models, performance evaluation and implementation. A model for performance evaluation should have time informations between the communicating entities to analyze average delay time, throughput, and so on. On the other hand, a model for validation should have logical informations to prove that there are no logical conflicts in procedure rules. From these different features, the designer has to develop two kinds of models to perform the performance evaluation and the logical analysis. This is a very tedious job.

This paper presents a framework which supports both performance evaluation and logical analysis within a unified formalism. Logical analysis techniques can be divided into two approachs, *single language approach* and *dual language approach.* Reachability analysis is a well known single language approach. It is practically impossible to perform complete analysis for complex systems[2]. On the other hand, the dual language approach uses two formalisms: *operational formalism* which describes the behavior of a system and *assertional formalism* which specifies the property of a system. We adopt Temporal Logic(TL) as an assertional language and DEVS formalism as a description language.

This paper is organized as follows. Section 2 describes the proposed framework for the dual language approach. In section 3, we describe the assertional language TL and its expansion procedure. And a projection mechanism for atomic DEVS models and a validation procedure are also described. We conclude this paper in section 4.

2 Proposed Framework

Figure 1 proposes a framework for logical analysis and performance evaluation within the unified DEVS formalism, where the logical analysis exploits the dual language approach. A modeler should develop DEVS models to perform performance evaluation. This is refined descriptions from informal requirements of a system. The DEVS formalism and the associated simulation algorithms provide sound modeling semantics and a simulation methodology[8]. Therefore the modeling and the performance evaluation processes are easily accomplished by using the DEVSim++[3].

0-8186-6440-1/94 $04.00 © 1994 IEEE

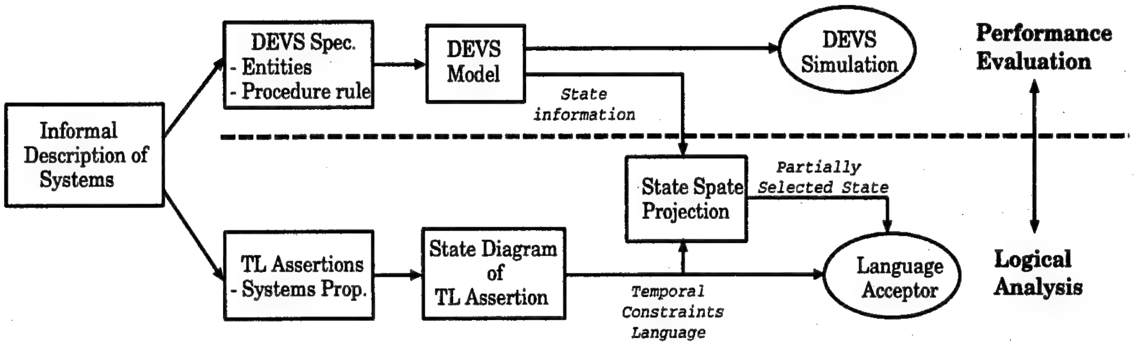


Figure 1: Overall System Configuration

The main advantage of the dual language approach is in its flexibility; the use of specification language provides a uniform notation for expressing a wide variety of correctness properties and it separate models from reachability assertions[6]. But the dual language approach also has the state explosion problem for complex systems. Therefore partial proof against given specifications is a reasonable solution to these problems[4]. The dual language approach with a projection mechanism can be an efficient method for the validation of large systems.

Temporal constraints that present temporal properties of the requirements are also required for logical analysis. These constraints are expressed by using Temporal Logic(TL). The temporal logic formula is translated into a finite state automaton. The state informations are obtained from the automata. These informations are applied to the DEVS model, developed for performance evaluation, to obtain a projected state space. Logical analysis is performed by using the projected state space and the finite automata of TL formulas.

3 Logical Analysis

The DEVS models for performance evaluation does not have any constraints about the desired states/events set and global state information of a system. For logical analysis, such information should be added to the DEVS models. The logical constraints which specify the state sequences of a system can be expressed in terms of temporal logic formulas. Timing information does not need for logical analysis. Thus, the time advance function fro an atomic DEVS model may not be used for logical analysis. The logical analysis is basically a searching procedure to find illegal states. For efficient analysis, there should be a mechanism to reduce the state space. We accomplish it by an extension of the DEVS formalism, *i.e*, add a projection function and state set informations derived from TL assertions into atomic DEVS models. A facility for global states manipulation is also added to coupled DEVS models.

3.1 Expression of Temporal Logic

Temporal logic assertions express sequence of states that should be satisfied during system execution. Therefore the state information of TL assertions can be used to project with respect to related states from the DEVS model of a target system. Temporal logic is an extended logic by adding the temporal operators to describe the timing relationship between entities. It has been widely used to specify discrete event systems such as concurrent system and communication protocol. The temporal operators and their meanings are as follows[1].

- $\Box(always)A$: A is true now and will always be true in the future.
- $\Diamond(sometimes)A$: A is true now or will be true sometimes in the future.
- $A\bigcup(until)B$: B is true now or A is true until B will be true.
- $\bigcirc(next)A$: A will be true next time.

To establish correspondence between TL assertions and a DEVS model, TL assertions are described by using the state variables and their values defined in the DEVS model. Let grammar G of the temporal expression be $\langle V_T, V_N, P, S\rangle$. Terminal V_T is a set of atomic formulas which contain no temporal operator. The non-terminal V_N follows the next operator $\bigcirc$. A given temporal expression, a non-terminal set and a

FINAL state constitute a states set S of the grammar. The production rules P for a set of temporal operators are as follows, where an expression in { } is a language accepted by the projection rules.

- $\Box A \Longrightarrow A \cdot \bigcirc(\Box A)\{A^*\}$
- $\Diamond A \Longrightarrow A \mid \neg A \cdot \bigcirc(\Diamond A)\{(\neg A)^* A\}$
- $A \bigcup B \Longrightarrow B \mid A \wedge \neg B \cdot \bigcirc(A \bigcup B))\{(A \wedge \neg B)^* D\}$
- $\neg \Box A \Longrightarrow \neg A \mid \bigcirc(\neg \Box A)\{(\neg A)^*\}$
- $\neg \Diamond A \Longrightarrow \neg A \mid A \cdot \bigcirc(\neg \Box A)\{A^* \neg A\}$
- $\neg(A \bigcup B) \Longrightarrow$
  $\neg B \mid B \wedge \neg A \cdot \bigcirc(\neg(A \bigcup B))\{(B \wedge \neg A)^* \neg B\}$

A temporal expression which describes a sequence of states is expanded to a current state condition and a next state condition using the grammar. This is based on the decision procedure in [7]. The expansion procedure for temporal expression is as follows.

(i) Start with application of the production rules shown above to TL assertions.

(ii) Set the non-terminal to a next state and the terminal to a transition condition. Any formula that contains only terminals becomes a transition condition for the **FINAL** state.

(iii) The outmost operator $\bigcirc$ for a next state is removed.

(iv) If a new state does not appears, then terminate. Otherwise go to (i).

After execution of the expansion procedure, a TL assertion is translated into a finite state automaton **R** as

$\mathbf{R} =< \mathbf{S}, \mathbf{A}, \delta_{\mathbf{R}} >$

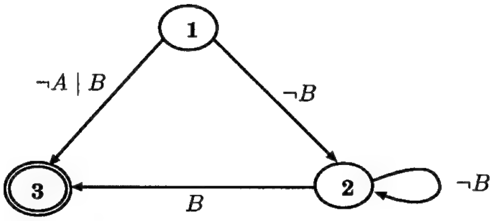
S : sequential states set;
A : logical assertions set;
δ : state transition function;
with the following constraints
S, A : finite set;
$\delta_R : S \times 2^A \rightarrow 2^S$;

In this paper, we use the alternating bit protocol(ABP) as an example system. A detailed description of the ABP appears in [5]. Consider the following property for the ABP: " *Error_free Transmission : If no transmission errors exist between Sender and Receiver, then messages are sent infinitely and they have alternate control bit*". The TL assertions of this property are as follows.

(i) $\Box(S.Error \wedge R.Error = false)$
(ii) $\Box((S.Phase = FM \wedge S.st = 0)$
$\quad \rightarrow \Diamond(R.Pahse = FA \wedge R.at = 0))$
(iii) $\Box((S.Phase = FM \wedge S.st = 1)$
$\quad \rightarrow \Diamond(R.Pahse = FA \wedge R.at = 1))$
(iv) $\Box((S.Phase = FM \wedge R.Phase = WM)$
$\quad \rightarrow (S.Phase = FM \wedge R.Phase = WM)$
$\quad \bigcup (S.Phase = WA \wedge R.Phase = FA))$
(v) $\Box((S.Phase = WA \wedge R.Phase = FA)$
$\quad \rightarrow (S.Phase = WA \wedge R.Phase = FA)$
$\quad \bigcup (S.Phase = FM \wedge R.Phase = WM))$

The model of the ABP should be correct if it satisfies the above TL assertions. To prove the property, we translate TL assertions into a finite state automaton. Figure 2 shows the resultant automata for the TL assertion (*iii*). The expansion procedure for this assertion is as follows. Let $S.Phase = FM \wedge S.at = 1$ be A, and $R.Phase = FA \wedge R.at = 1$ be B. The following is the expansion procedure for the given property.

1st step : apply expansion rule for $\Diamond B$
$\quad \neg A \mid \Diamond B = \neg A \mid B \mid \neg B \cdot \bigcirc(\Diamond B)$
2nd step : determine transition conditions
$\quad$ transition to **FINAL** state : $\neg A \mid B$
$\quad$ transition to next state($\Diamond B$) : $\neg B$
3th step : apply expansion rules to $\Diamond B$
$\quad \Diamond B = B \mid \neg B \cdot \bigcirc(\Diamond B)$
4th step : determine transition conditions
$\quad$ transition to **FINAL** state : B
$\quad$ transition to next state($\Diamond B$) : $\neg B$
5th step : ($\Diamond B$) appears again : stop expansion



$A = (S.Phase = FM \wedge S.st = 1)$

$B = (R.Phase = FA \wedge R.at = 1)$

Figure 2: Finite State Automaton

The state values set which is related to the given properties can be extracted from the transition condition $\mathbf{2^A}$ of **R**. This is done by the grouping the values set used as transition conditions. The state variables of **SENDER** and **RECEIVER** for the above property can be grouped as follows: $S.Phase = \{\{WA\}\}$, $S.st = \{\{0\}\}$, $S.Error = \{\ \}$, $R.Phase = \{FA, \{WA\}\}$, $R.at = \{1, \{0\}\}$ and $R.Error = \{\ \}$. Therefore, states that have the same values except for *Error* can be treated as an equivalent state. Table 1 represents the values set of state variables of the ABP ofr the *Error_free Transmission* property.

Spec #	sv	SENDER	RECEIVER
1	Error	false	false
2	Phase	{WA}	FA, {WM}
	st	{1}	—
	at	—	0, {1}
3	Phase	{WA}	FA, {WM}
	st	{0}	—
	at	—	1, {0}
4	Phase	{WA}, WA, FM	FA, {FA}, WM
5	Phase	FM, {FM}, WA	{WA}, WM, FA

Table 1. Grouped state variables for TL assertions

3.2 Projection Procedure

The projection is an efficient method to reduce the space/time complexity of logical analysis. Assume that a model supports various properties and some of them are disjoint. Then TL assertions for a certain property use only a subset of domain of a state variable. Therefore the states which have the unused values can be grouped in one state.

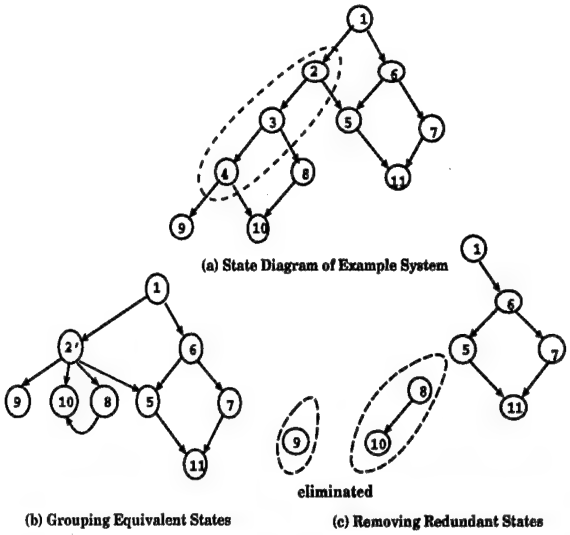
Definition. 1 *Let $M(S_i)$ be a state variable of an atomic DEVS model and $M(V_i)$ be domain of $M(S_i)$. Then the set of states of the model is $S_M = M(V_1) \times M(V_2) \times \cdots M(V_n)$.*

Definition. 2 *Let $R(S_i)$ be a state variable that is used as transition conditions in the automata* **R** *and $R(V_i)$ be a set of grouped value set. Then the set of states of requirements is $S_R = R(V_1) \times R(V_2) \times \cdots R(V_n)$.*

Definition. 3 *Projection is a mapping of a set of states in S_M into a state in S_R based on $R(V_s)$.*
(i) The states in S_M which contain a value in the used value set $\bigcup_{1 \leq i} R(V_i)$ is aggregated into a state in S_R. (ii) The states in S_M which contain values that does not appear in the used value set $\bigcup_{1 \leq i} R(V_i)$ are removed from S_M. The resultant isolated states are also removed. (iii) If i state variables do not used in the assertions, then the n dimensional state space is mapped into the $n - i$ dimensional image.

Consider a system shown in Figure 3 (a) that is described by 2 state variables $v_1 = \{true, false\}$, $v_2 = \{n \mid n \geq 1\}$. Let states s_2, s_3 and s_4 be $s_2 = \{v_1 = false, v_2 = 2\}$, $s_3 = \{v_1 = false, v_2 = 3\}$ and $s_3 = \{v_1 = false, v_2 = 3\}$. Assume that state transition conditions which are obtained from the expansion procedure have a value group $v_2 = \{1, \{2, 3, 4\}\}$. If a transition condition for TL assertions satisfies $v_1 = false$, then the system can transit to these states. Therefore these states are equivalent and can be grouped. Figure 3 (b) shows the results after grouping.

If TL assertions do not use the value $v_1 = false$, then the system never transit to the states that contain this value. So, these states can be removed from the original system. Figure 3 (c) shows the result after removing those states.



(a) State Diagram of Example System

(b) Grouping Equivalent States

(c) Removing Redundant States

Figure 3: Projection of Example System

The DEVS formalism is extended for logical analysis and projection. The projection function is a mapping of states from an atomic DEVS model to a state in a finite state automaton of a TL assertion. S_R is the states set that is equivalent to the grouped states of the DEVS model. Formally, the specification of an atomic mode M is as follows.

$$\mathbf{M} =< \mathbf{X}, \mathbf{S_M}, \mathbf{Y}, \delta_{\mathbf{int}}, \delta_{\mathbf{ext}}, \lambda, \mathbf{ta}, \mathbf{f_R}, \mathbf{S_R} >$$

X : input events set;
S_M : sequential states set;
Y : output events set;
δ_{int} : internal transition function;
δ_{ext} : external transition function;

λ : output function;
ta : time advance function;
f_R : projection function for requitements;
S_R : states set of requirements;

with the following constraints,

X, Y, S_M : infinite but countable set;
$\delta_{int} : S_M \rightarrow S_M$;
$\delta_{ext} : Q \times X \rightarrow S_M$;
$Q = \{(s,e) \mid s \in S_M, 0 \leq e \leq ta(s)\}$;
Q: total state of M,
e: elapsed time after scheduling;
$\lambda : S_M \rightarrow Y$;
$ta : S_M \rightarrow Real$;
$f_R : 2^{S_M} \rightarrow S_R$;

To validate, the gathering and tracking facilities of global states of the coupled DEVS model are required. So a means for manipulating the global states set is added to the coupled DEVS formalism.

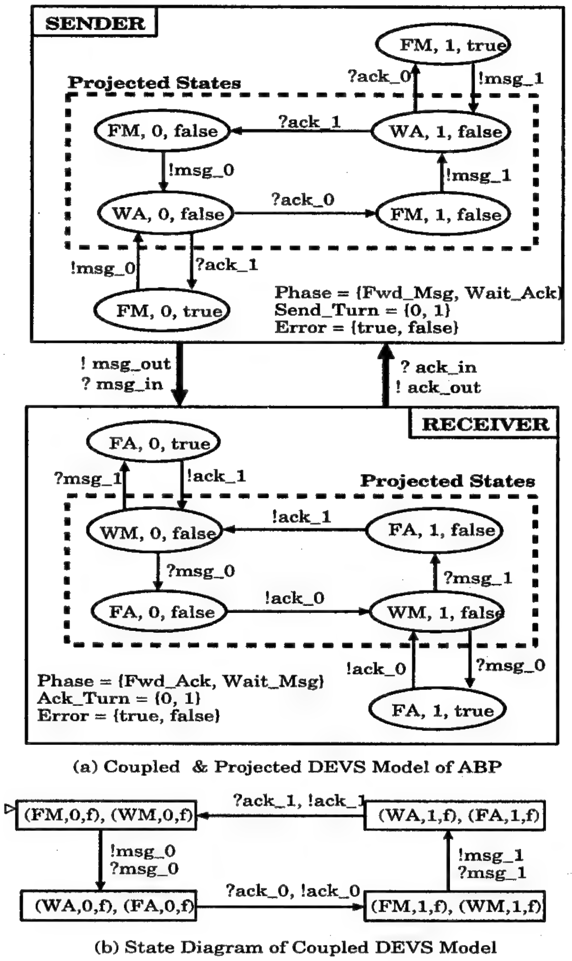
$$\mathbf{DN} = < \mathbf{D}, \{\mathbf{M_i}\}, \{\mathbf{I_i}\}, \{\mathbf{Z_{i,j}}\}, \mathbf{SELECT}, \mathbf{S_G}, \delta_{\mathbf{DN}} >$$

D : component names set;
for each i in D,
M_i : DEVS component i in D;
I_i : set of influencees of i;
for each j in I_i,
$Z_{i,j} : Y_i \rightarrow X_j$
: i-to-j output translation function;
$SELECT : 2^D \rightarrow D$: tie-breaking selector;
$S_G : \times S_{R_i}$: global states set;
$\delta_{DN} : S_G \times 2^A \rightarrow 2^{S_G}$: state transition function;

Figure 4 (a) shows state diagrams of the atomic models SENDER and RECEIVER. The global state diagram for the projected atomic models is shown in (b). The dotted area presents the projected states with respect to the property *Error_free Transmission*. The global state diagram obtained from the projected atomic models is equivalent to the projection result of the original coupled model. Therefore application of the projection on the atomic DEVS model is more efficient because the state space complexity is reduced.

3.3 Validation Procedure

The logical analysis is performed by an acceptance checking : check whether a TL assertion accepts the state transition sequences of a coupled DEVS model. If the model is correct, then a sequence of states in a cycle of the model would be accepted by the temporal constraints language. The validation algorithm is as follows.



(a) Coupled & Projected DEVS Model of ABP

(b) State Diagram of Coupled DEVS Model

Figure 4: Global State Diagram of Projected ABP

Validation Algorithm

```
Var
    Stack: stack of G;
    $g_i$ : global states set : $S_i \times S_{G_i}$;
    $S_n$ : next states set of FSA;
    $t_n$ : transition condition of FSA;
    $S_{G_n}$ : next states set of MODEL;

begin
    Stack := { };
    $g_i := g_0$;
    push($g_0$, Stack);
    while not_empty(Stack) do begin
        $S_n := \delta(S_i, t_i)$;
        $S_{G_n} := \delta_{DN}(S_{G_i}, t_i)$;
        if $S_{G_n} = \{\ \}$ then
            go to Label;
        $g_n := S_n \times S_{G_n}$;
        for $\forall g_n$ do
            if ($g_n \notin$ Stack) then
                push($g_n$, Stack);
            else if acceptance_check($g_n$) = true then
                terminate(model is correct);
        end for;
        Label : $g_i$ = pop(Stack);
    end while;
terminate(conflict exist);
end;
```

By using the above algorithm, we can find a loop from the transitions of a FSA(Finite State Automaton) and a DEVS model in the Figures 2 and 4. The possible next states $g_1([2,((WA,0,f),(FA,0,f))])$ and $g_2([3,((WA,0,f),(FA,0,f))])$ are obtained from the initial $state g_0([1,((FM,0,f),(WM,0,f))])$. Next state of g_2 becomes $g_3([1,((WA,0,f),(FA,0,f))])$ although g_2 is an acceptance state. This is because a cycle of sequences of global states does not exists. And g_1 transit into $g_4([2,((WA,0,f),(FA,0,f))])$ because next state of the model can accept the transition condition of the FSA. The state transition of the model based on transition condition of the FSA is made until a cycle of state sequences of the model is detected and the FSA reaches the acceptance state. If a model has a deadlock, then it can not proceed at that state. Therefore the model can not reach the acceptance state until the algorithm is terminated. The validation algorithm can also be applied to the negation of a TL assertion. Such negation of a given TL assertion may increase validation speed for some cases.

4 Conclusion

This paper presents a unified framework for performance evaluation and logical analysis within the DEVS formalism. A performance analysis model has timing informations between the communicating entities. That is used to analyze time related performance such as average delay time and throughput, *etc.* A validation model has logical informations which is used to prove systems properties or procedure rules. This is accomplished by an extension of the DEVS formalism.

To solve the state space explosion problem during the validation phase, we exploit a projection mechanism using external TL assertions. This is a very efficient method because state reduction is made before the atomic models are coupled. Then various validation techniques which are used in the dual language approach can be applied.

Acknowledgement

The authors would like to thank ETRI(Electronics and Telecommunications Research Institute) for supporting this research. This research was done in the context of the ETRI project on ATM network performance study.

References

[1] Reinhard Gotzhein, " Temporal logic and applications - a tutorial," *Computer Networks and ISDN Systems 24*, 1992.

[2] Gerard J. Holzmann, *Design and Validation of Computer Protocols*, Prentice-Hall, Inc., 1991.

[3] Tag G. Kim, "DEVSIM++ User's Manual: C++ Based Simulation with Hierarchical Modular DEVS Models", Computer Engineering Lab., Dept. of Electrical Engineering, KAIST, 1994.

[4] Simon S. Lam, A. Udaya Shankar, "Protocol Verification via Projections", *IEEE Transactions on Software Engineering*, Vol. SE-10, No. 4, July 1984,

[5] Jawahar Malhotra, Scott A. Smolka, Alessandro Giacalone, Robert Shapiro, "Winston : A Tool for Hierarchical Design and Simulation of Concurrent Systems, "

[6] Jonathan S. Ostroff, *Temporal Logic for Real-Time Systems, Advanced Software Development Series. 1*, Research Studies Press Ltd., 1989.

[7] Pierre Wolper, "Temporal logic can be more expressive," *22nd Annual Symposium on Foundation of Computer Science*, pp.340-348, 1981.

[8] B.P. Zeigler, *Multifacetted Modeling and Discrete Event Simulation:* Academic Press, Orlando, FL, 1984.

Session 2B:

DEVS Formalism: Modeling Methodology

Distributing and Maintaining Knowledge: Agents in Variable Structure Environments

A.M. Uhrmacher R. Arnold

Department of Computer Science
Institute of Artificial Intelligence
D-89069 Ulm
Germany

Abstract

The maintaining and adaptation of knowledge within changing environments is one of the crucial aspects in decentralized controlled and distributed systems. To explore different strategies and their consequences, we use an example where processors, structured in a hierarchy, are hired or fired responding to the requests of the current work load. The modeling and simulation approach uses the actor metaphor of open systems, where the nodes of the hierarchy are perceived as autonomous agents with an internal explicit model about their environment. Questions about the distribution and maintenance of knowledge referring to the structure of systems, needs for cooperation, and the change of roles are discussed against the background of the example and complete the picture about the specific effects of the single strategies. DEVS, a knowledge-based simulation environment, constitutes the background of our exploration.

1 Introduction

To function autonomously in a structural changing environment entities need knowledge about their surroundings and their own role in this environment. To describe those entities and their behavior, we equipped an object-oriented modeling scheme with internal explicit models about the structure, i.e. the composition and coupling, of their environment, thus following system-theoretic approaches [6]. Assuming not only one single entity works autonomously in an environment but a group of entities, the questions arise how much knowledge each entity has to possess in order to guarantee the functioning of the whole system and how the local internal models are related to each other in a dynamically evolving environment with changing structures. With those questions we approach the area of open systems where bounded knowledge and bounded influences are discussed in the context of decentralized control [2].

The knowledge that is required by the single agents depends typically on the problem that is tackled with the distributed systems. Often, the desired functionality can be achieved based on concepts of self-organization which obviate detailed knowledge and reasoning capabilities of the involved agents [4]. Besides the problem itself, the ability of the agents involved in problem solving and the perspective which shall be pinpointed by modeling influence the internal model of agents.

Analyzing different scenarios of a small example the adaptation of local views will be discussed based on the concept of endomorphic intelligent agents, which has been developed to deal with structural changes in DEVS.

2 Internal Models in DEVS

In an extension of DEVS, the perception of agents which may refer to the environment, to the agents themselves and the existing interrelationships is represented as internal models [6]. They realize the awareness about the agent's own embedding in the network of communication [1], and represent explicit models within models, which Zeigler called "Endomorphy" [10]. Actions are based on and directed to these internal models. Thus a constructivistic view is supported as the world is changed corresponding to the local views of the agents.

During changes in the environment internal models have to be adapted frequently. The adaptation can

0-8186-6440-1/94 $04.00 © 1994 IEEE

be initiated by communication with other agents or by discovering inconsistencies between the local model and the external world by observing. In the context of this paper we will concentrate on the former.

Typically models in the discrete event simulation system, DEVS, are defined either as atomic models or as coupled models [10]. The behavior of a coupled model is completely determined by the behavior of the atomic models. This reductionistic modeling approach inhibits some problems for the realization of autonomy [6]. To overcome this difficulties, a special kind of endomorphic agents with structural knowledge about themselves and their environment has been developed. Endomorphic intelligent agents responsible for structural change (M_{esa}) are described as atomic models in DEVS whose definition is extended by an internal model (IMS).

$$M_{esa} =_{df} \langle X, S, IMS, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$$

X is the set of input ports for receiving external input events, S is the sequential state set, Y is the set of output ports for sending the generated outputs as external events, δ_{int} (δ_{ext}) is the internal, respectively external, transition function dictating state transitions due to internal (external inputs) events, λ is the output function which generates events as outputs, and ta is the time advance function. The internal model IMS expands the state of the atomic model capturing the structure of the outside world it is controlling. This information about the structure of the outside world is encoded as a set of abstract structure models AS.

$$\begin{aligned} IMS(a) &=_{df} \{ AS(m,\ a)^+ \} \\ AS(m,a) &=_{df} \\ &\langle\ RM(m,a),\ X(m,a),\ S(m,a),\ Y(m,a), \\ &CM_{Names}(m,a),\ CM_{AS}(m,a),\ C(m,a)\rangle \end{aligned}$$

Each abstract structure model comprises the root-model RM, the model on the highest organization level the agent is controlling, its input ports X, its state S, its output ports Y, the names of its components CM_{Names}, which can also be described as abstract structure models themselves CM_{AS}, and the coupling that exists among them C. The abstract structure model depends not only on the model m that is controlled but also on the model that is controlling, i.e., the endomorphic intelligent agent a. In the abstract structure model the reductionistic view is exchanged against a more holistic view. Influenced by the modeling system EMSY [5], it allows to attribute a state and in a future version also rules to the internal model. With the latter the agents will be able to reason about the behavior of atomic and coupled models as well [7, 3]. Abstract structure models do not distinguish between atomic and coupled models.

3 Hiring and Firing

To discuss some phenomena in distributing and maintaining knowledge within a group of endomorphic intelligent agents, we will use the "hiring-firing" example Zeigler introduced in 1989 [9]. Processors are structured in a hierarchy where the internal nodes are called managers and the leaves workers. Depending on the work load, the processor tree will expand or shrink. The first question is who decides the hiring and firing process. The answer is directly related to the problem which of the nodes are considered to be intelligent. The other question is who is hired and who is fired. The latter effects the dynamics within the tree and the role changes that the nodes have to undergo.

Different strategies are possible to hire and fire nodes in the processor tree:

- Hiring:
    1. Hiring a parent between the leaf and its former parent.
    2. Transforming the leaf to an internal node and creating a new leaf.
- Firing :
    1. Firing the parent and replacing the position by the leaf.
    2. Firing the leaf and transforming the parent into a leaf.

Hiring 1. in combination with firing 1. implies no role changes. The entities are staying workers or managers through their whole life, workers are living longer in the hierarchy while their distance to the top manager as well as their managers vary frequently. Scenario hiring 2. in combination with firing 1. has the effect that by hiring leaves, the former workers, become managers and new workers are hired, while in the phase of firing the managers are the first that have to leave.

4 Realization in DEVS

Different possibilities exist to realize the problem in DEVS. Each pinpoints a different perception of the

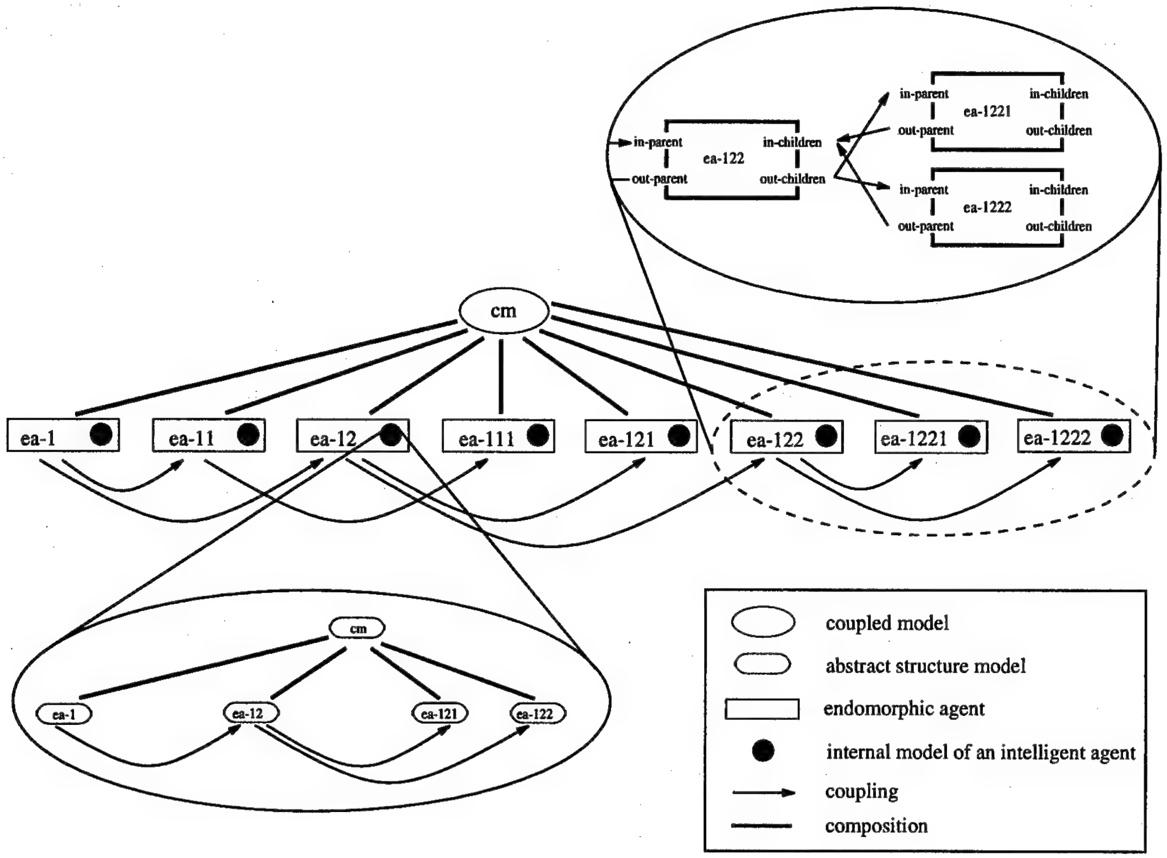


Figure 1: Processor Hierarchy Realized by Coupling - "Hiring Workers and Firing Managers" Scenario

problem. One of those possibilities is constituted by a "flat" representation where all nodes are considered to be intelligent with the ability to adapt easily to different roles.

4.1 A Flat Representation

All the processors are described as intelligent agents and as components of one coupled parent (Fig. 1). Thus, all nodes of the hierarchy are able to process jobs, and the hierarchy is constructed by coupling. The resulting model hierarchy is from the compositional point of view flat. Each agent has output ports and input ports to communicate with its "coupling-parent" and its "coupling-children". While the parent port is only connected to at most one other agent, the children port can be connected to several agents. In all scenarios the internal model will typically refer only to a certain part of the entire processor tree, thereby establishing a local view of the agent.

In scenario "Hiring Workers and Firing Managers" the internal model of the agents includes its parent, its children, itself, and the existing interrelations. As an agent is only allowed as a worker to hire other workers, it has to know if it has children of its own. The firing of parents necessitates the information about the parent because during the process of firing the parent might be substituted by the leaf. As the parent is intelligent itself, a message sent to the parent with the requirement to fire itself and substitute its position by the leaf, leads to a test if there exist any sisters. In case no sister exist, the parent initiates the structural change itself. To substitute the couplings correctly it is necessary for the parent to know its own parent. Therefore, the information about the parent is needed, same as the cooperation of the parent.

In addition to knowledge about its parent, itself and its children, the internal model includes also knowledge about the phases of children. As a job reaches a processor only by passing the parent and each completion is reported to the parent, parents can easily keep track of their children's activities. Those phases con-

$$
\begin{aligned}
M_{esa}(ea\text{-}12) \;=\; \langle & X = \{\mathit{in\text{-}parent},\ \mathit{in\text{-}children}\},\\
& S = \{\mathit{Phase},\ \ldots\},\\
& IMS = \{AS(cm,\ ea\text{-}12)\},\\
& Y = \{\mathit{out\text{-}parent},\ \mathit{out\text{-}children}\}, \ldots \rangle
\end{aligned}
$$

where

$$
\begin{aligned}
& AS(cm,\ ea\text{-}12) = \\
& \quad \langle RM = cm,\\
& \quad\ X = \{In\},\\
& \quad\ Y = \{Out\},\\
& \quad\ CM_{Names} = \{ea\text{-}1, ea\text{-}12, ea\text{-}121,\ ea\text{-}122\}\\
& \quad\ CM_{AS} = \{\langle RM = ea\text{-}1,\ X = \{\mathit{in\text{-}parent}, \mathit{in\text{-}children}\}, Y = \{\mathit{out\text{-}parent},\ \mathit{out\text{-}children}\}\rangle,\\
& \qquad\qquad \langle RM = ea\text{-}12, X = \{\mathit{in\text{-}parent},\ \mathit{in\text{-}children}\},\ Y = \{\mathit{out\text{-}parent},\ \mathit{out\text{-}children}\}\rangle,\\
& \qquad\qquad \langle RM = ea\text{-}121,\ X = \{\mathit{in\text{-}parent},\ \mathit{in\text{-}children}\},\ Y = \{\mathit{out\text{-}parent},\ \mathit{out\text{-}children}\}\rangle,\\
& \qquad\qquad \langle RM = ea\text{-}122,\ X = \{\mathit{in\text{-}parent},\ \mathit{in\text{-}children}\},\ Y = \{\mathit{out\text{-}parent}, \mathit{out\text{-}children}\}\rangle\},\\
& \quad\ C = \{(ea\text{-}12.\mathit{out\text{-}parent}\{ea\text{-}1.\mathit{in\text{-}children}\}),\ (ea\text{-}12.\mathit{in\text{-}parent}\{ea\text{-}1.\mathit{out\text{-}children}\}),\\
& \qquad\quad (ea\text{-}12.\mathit{out\text{-}children}\{ea\text{-}121.\mathit{in\text{-}parent}, ea\text{-}122.\mathit{in\text{-}parent}\}),\\
& \qquad\quad (ea\text{-}12.\mathit{in\text{-}children}\ \{ea\text{-}121.\mathit{out\text{-}parent}, ea\text{-}122.\mathit{out\text{-}parent}\})\}\rangle
\end{aligned}
$$

Figure 2: Extract of the Internal Model - "Hiring Workers and Firing Managers" Scenario

tain the information whether the subtrees whose roots are represented by the children are entirely occupied or possess at least one passive node; consequently, they may not correspond to the actual phases of the children themselves, i.e. the *phase* : *passive* in the abstract structure model of a child implies not necessarily its idleness.

According to the internal model, each processor directs the jobs to those children with free capacities. Actually, the job is sent to all children (Fig. 1), but each child decides whether it can ignore the message based on the information which accompanies the message: the name of the addressee.

When all children, according to the internal model, and the processor itself are active, and a job arrives, the job is passed randomly down the hierarchy until it reaches a leaf. As the leaf is active itself, it decides to create a worker and to become a manager (Fig. 3). Hence, leaves are only created when all nodes are busy. Yet, the strategy does not guarantee a balanced growth of the tree.

The firing of the managers is initiated by a longer phase of passiveness by one of the workers, and is employed by an internal transition function. We stated before, that based on the assumptions that the managers are willing to cooperate and all agents have the same kind of knowledge and abilities, the agents need to know only about themselves, their children, their parent and the coupling that exist among them. In our model the manager is willing to cooperate, i.e. to put the worker in his position and fire himself, except he himself is active or other sisters exist; in the latter case the manager will simply remove the leaf and stay in his position.

If workers had to initiate the firing of their managers without any cooperation, the internal model would have to include the knowledge about sisters and about grandparents as well. In our example the internal model does not include this information, therefore the firing of managers requires their cooperation to implement the structural change and to update the internal models of the effected agents correctly. In our example the decision of hiring and the decision of firing are initiated locally based on the internal models of the agents. Depending on the strategies how we resolve the "hiring and firing" question the internal model will reflect a different view of the world.

Zeigler [9] proposed a "Hiring and Firing of Workers" in a flat representation where the firing should be initiated and implemented by the managers, whereas the hiring should be the task of the workers. Obviously, this scenario would require the information about children and grandchildren while the information about the parent would become superfluous. After we discussed the "Hiring Workers and Firing Managers" scenario in some detail it is easy to fathom how an implementation of the "Hiring and Firing Workers" scenario would look like.

```
; when ea receives (job job-name processing-time ea-name) on port in-parent

(let* ((self-name (get-name ea))
       (internal-model (get-internal-model ea)))

  (if (equal ea-name self-name)
      (if (equal (get-phase ea) 'busy)
          (if (> MAX-CHILDREN
                 (number-of-children self-name internal-model))
              (let* ((new-name (create-name 'ea))
                     (new-ea (make-endomorphicmodel
                              new-name
                              :internal-model
                              (make-abstract-structuremodel
                               :root-model (get-root-model internal-model)
                               :components (self-name new-name)
                               :coupling (make-coupling
                                          :ic (((self-name out-children)(new-name in-parent))
                                               ((new-name out-parent)(self-name in-children))))))))
                ...
                (add-component internal-model new-name)
                (add-abst-model internal-model
                                (make-abstract-structuremodel
                                   :root-model new-name
                                   :state '((phase busy))))
                (add-coupling internal-model
                              (make-coupling
                                 :ic  (((self out-children)(new-name in-parent))
                                       ((new-name out-parent)(self in-children)))))

;         else (number of children = MAX-CHILDREN)  send job to one of your children randomly

;     else (phase = passive) process job

; else (ea-name <> your name) ignore external event
```

Figure 3: Extract of the External Transition Function Responsible for Hiring

The internal model depends not only on the selected strategy of hiring and firing, but also on the distribution of "intelligence" within the network of nodes. The internal model will naturally be different if only a certain group of agents possesses an internal model about their environment and themselves.

4.2 A Compositional Representation

If only workers of the hierarchy are considered to be intelligent, the internal model has to reflect the whole path from the worker up to the root manager. If we can guarantee that intelligent coordinators exist on each hierarchical level, it will be sufficient to capture only one level up and one level down the hierarchy. However, both scenarios suggest to leave the flat representation and to construct the hierarchy of processors by composition, describing managers as coupled models.

In the following, we will discuss the hiring and firing of processors based on intelligent leaves, assuming no particular arrangement of intelligent leaves and coupled models within the tree. The processor hierarchy is built by composition representing the internal nodes as coupled models. Following a reductionistic approach, coupled models in DEVS have no activity of their own. They are constituted by input and output ports (X, Y), a set of components (M), a coupling structure (C) and a function to select the component ($Select$) that is allowed to produce its next event ([10]).

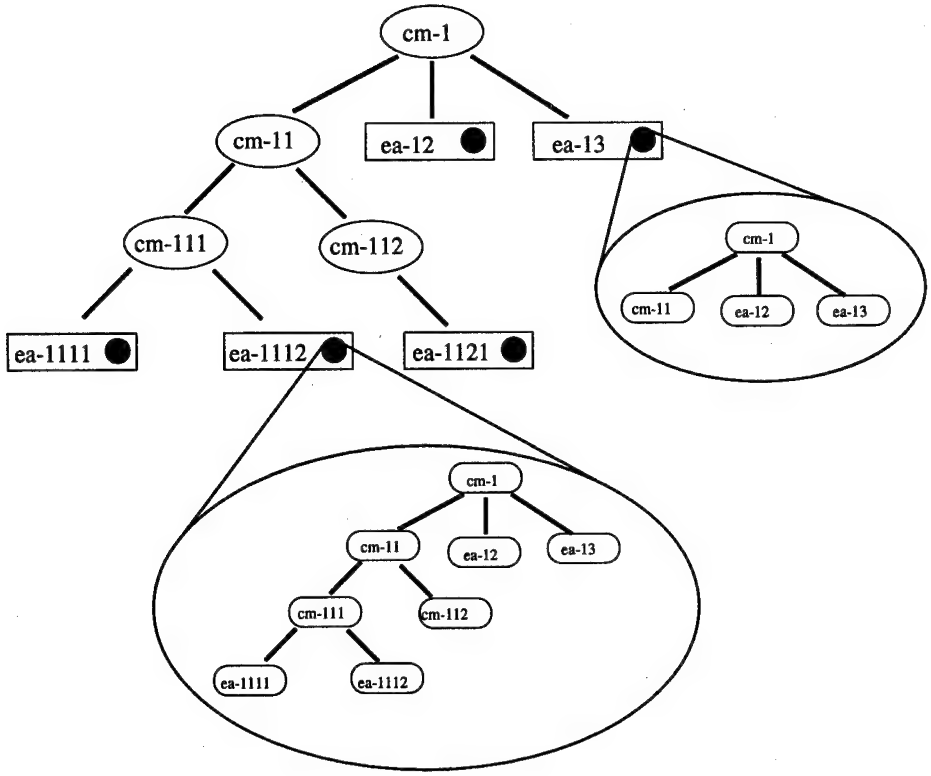


Figure 4: Processor Hierarchy Realized by Composition - "Hiring and Firing of Managers" Scenario

$$CM =_{df} \langle X,\ Y,\ M,\ C,\ Select\rangle$$

Thereby, the role of managers is reduced to distribute the work load to the next lower level of the hierarchy and to structure its communication. Managers have no ability to decide or to realize anything, neither hiring nor firing. Thus, the full responsibility for changing structures and processing jobs is taken by the workers. This implies that all the information for structural changes has to be located within the leaves independently of the strategy we choose for hiring and firing.

The latter will only influence the dynamics within the processor tree and the degree of role changes the single nodes have to undergo. Role changes are obviously more cost intensive than they are in the context of the flat representation. In the prior example all nodes of the processor tree had the ability to hire and fire on principle. Whereas in this case a role change means to turn an endomorphic intelligent agent into a coupled model and vice versa. Only the name remains as a sign of its identity; yet its internal structure would change dramatically. Therefore, we choose a strategy that avoids role changes: parents are hired and placed between the former parent and the leaf, parents are fired and replaced by the leaf, if no other sisters exist.

If we do not assume any particular distribution of intelligent leaves, the internal models of the agents have to cover the entire path from the top manager downwards (Fig. 4). As the single agent can not rely on the informations of other agents, a successive firing of managers requires to know the path from the top manager down to the leaf. The information of possible sisters is important to decide whether the parent can be fired and substituted or whether the leaf is the one to be fired itself.

Same as in the "flat" example, the initiating and realization of structural changes, and the communication about it can be implemented by a combination of external, internal and output function. Those functions are only based on the information which is locally available in the model. Thus, our approach fits nicely into the general framework of DEVS, where the transition and output functions are based on the locally available information, and where the communication between models takes place via the output function and the output ports, only. Unlike other approaches [8] it obviates the need for additional constructs which might effect the locality and accessibility of information. This advantage is paid for by keeping and maintaining information redundantly in the internal models.

5 Conclusions

Based on the internal local perception of the world, agents in DEVS are able to handle flexibly variable structure environments. Different strategies in invoking structural changes can be implemented easily based on the internal models, in which knowledge about the structure of systems, their composition and coupling, is expressed explicitly. Based on this concept and a few scenarios, we could illustrate some of the specific phenomena in distributing and maintaining knowledge between autonomous agents. The more intelligent agents exist, the less important is a global view of the world because the local views of different agents have a complementary effect. Even if all agents are considered to be intelligent the need for cooperation and communication can be reduced by expanding the internal model of agents.

Organizing knowledge about the structure of systems in decentralized internal models leads necessarily to redundancies and inconsistencies. Therefore, like in other open systems, reasoning and coordinated action depend on using debate and negotiation to mediate between the local views of the involved agents. It will be the subject of further research to provide suitable pattern to resolve global inconsistencies for modeling endomorphic intelligent agents in variable structure environment.

Acknowledgment

Most of the ideas presented in this paper have been developed during the stay of the first author as an visiting scholar of the Alexander von Humboldt Foundation at the Department of Electrical and Computer Engineering at the University of Arizona in Tucson. We would like to thank Dr. Bernard Ziegler for many hours of constructive discussions without which this paper would not have been possible.

References

[1] Durfee E.H., Lesser V.R., Corkill D.D., 1987: Cooperation through Communication in a Distributed Problem Solving Network. In: Huhns M.N. (ed.): Distributed Artificial Intelligence. Morgan Kaufman, San Mateo., 29-58.

[2] Gasser L., 1991: Social Conceptions of Knowledge and Action: DAI Foundations and Open Systems Semantics. Artificial Intelligence 47, 107-138.

[3] Praehofer H., Bichler P., Zeigler B.P., 1993: Synthesis of Endomorphic Models for Event-Based Intelligent Control Employing Combined Discrete/Continuous Simulation. In: AI, Simulation, and Planning in High Autonomy Systems, Proc. 4. Conference, IEEE Computer Society Press, San Diego, 120-127.

[4] Steels L., 1990: Cooperation between Distributed Agents through Self-Organization. In: Demanzeau Y. and Müller J.-P., (eds.): Decentralized AI. Elsevier Publishers B.V., North Holland, 175-195.

[5] Uhrmacher A.M., 1994: Reasoning about Changing Structure: A Modeling Concept for Ecological Systems. International Journal on Applied Artificial Intelligence. (To appear).

[6] Uhrmacher A.M., Zeigler B.P., 1994: Variable Structure Models in Object-Oriented Simulation. International Journal on General Systems. (To appear).

[7] Wesson R.B., Hayes-Roth F.A., Burge J.W., Stasz C., Sunshine C.A., 1981: Network Structures for Distributed Situation Assessment." IEEE Trans. Systems, Man and Cybernetics, Vol. SMC-11, 5-23.

[8] Zeigler B.P. and Praehofer H., 1989: Systems Theory Challenges in the Simulation of Variable Structure and Intelligent Systems." In: Computer Aided Systems Theory - Lecture Notes. Springer, Berlin.

[9] Zeigler B.P., 1989: Concepts for Distributed Knowledge Maintenance in Variable Structure Models. In: Elzas M.S., ren T.I., and Zeigler B.P. (eds.): Modeling and Simulation Methodology - Knowledge Systems' Paradigms. Science Publishers. (North Holland), 45-54.

[10] Zeigler B.P., 1990: "Object-Oriented Simulation with Hierarchical, Modular Models - Intelligent Agents and Endomorphic Systems". Academic Press, San Diego.

Variable DEVS - Variable Structure Modeling Formalism: An Adaptive Computer Architecture Application

Fernando J. Barros and Maria T. Mendes
Laboratório de Informática e Sistemas
Universidade de Coimbra
Urb. Quinta da Boavista Lote 1, 1°
P-3000 Coimbra, Portugal

Bernard P. Zeigler
Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721

Abstract

Conventional modeling theory gives support only for representing model behavior, providing little aid for describing changes in model structure. Some models are better represented by changes in their structure. Instead of forcing this changes to be represented at the simple behavioral level, a strong theoretical support is needed to allow the representation of structural changes in a natural way. In this paper we present a modeling methodology for representing variable structure systems. Examples of such systems include adaptive computer architectures, ecological systems, fault tolerating computers. We describe an application of this methodology to the modeling and simulation of an adaptive computer architecture.

1 Introduction

A variable structure model can transform itself in a model of a variant family. Examples of applications of systems which exhibits structural changes are: Reconfigurable computer architectures [1], [2], [3], [4], [5], fault tolerance computers [6] and ecological systems [7]. There is currently little support for variable structure modeling.

Several approaches have been proposed to a methodology of variable structure modeling. A multilevel system is described in [8]. In this hierarchical system the first level represents the conventional behavioral model where the simulation occurs. The second level controls the structure of the first level. The structure of second level can be changed by the third level of this multilevel hierarchy.

Changes in structure are currently supported by controlled-models. In these models there are fixed connections between the controller element and the other components. Is thus possible to insert or delete components in controlled models due to the automatic handling of connections [9].

A broad discussion of variable structure models is presented in [10]. This approach is based upon endomorphic agents. These intelligent agents keep an internal representation of model structure. Endomorphic agents start changing their own internal representation of the system using only the external and internal functions provided by DEVS definition of atomic models, preserving thus modularity constraints of DEVS formalism. However, the automatic mapping from changes in model representation to changes in the models themselves is a topic of current research.

A coupling formalism called *name-directed coupling* [11], can handle changes in connections. However this formalism does not currently provide support for add/delete model operations.

This paper focuses the extension of DEVS formalism with variable structure constructs. We extend DEVS to support a higher level of control that is able to control the behavior of the standard simulation level. This higher layer is compatible with modularity model building. To

ISBN 0-8186-6440-1. Copyright © 1994 IEEE. All rights reserved.

0-8186-6440-1/94 $04.00 © 1994 IEEE

illustrate these concepts we present as an apllication an adaptive computer architecture.

2 Review of DEVS formalism

DEVS formalism was introduced by Zeigler and is a systems theory tool for describe discrete event systems. In DEVS formalism is necessary to define basic models and how these models are connected. An atomic model is defined by the 7-tuple:

$M \equiv <X,S,Y,\delta_{int},\delta_{ext},\lambda,ta>$, where
$X \equiv$ set of external input events;
$S \equiv$ set of sequential states;
$Y \equiv$ set of output events;
$\delta_{int}: S \rightarrow S \equiv$ internal transition function;
$\delta_{ext}: Q \times X \rightarrow S \equiv$ external transition function;
where $Q = \{(s,e) \mid s \in S, 0 \leq e \leq ta(s)\} \equiv$ total state set;
$\lambda: S \rightarrow Y \equiv$ output function;
$ta: S \rightarrow R_0^+ \equiv$ time advance function.

Atomic models can be connected to form coupled models. Coupled models are defined in DEVS formalism by the 5-tuple:

$CM \equiv <X,Y,M,C,select>$, where
$X \equiv$ set of external ports;
$Y \equiv$ set of internal ports;
$M \equiv$ set of components;
$C \equiv$ connections between components;
$select \equiv$ tie break selector.

DEVS provides a powerful mechanism for specifying hierarchical and modular models. Large models can be built using atomic models, and these new coupled models can be used as building blocks to make more complex ones. Full description of DEVS formalism can be found in [12], [13].

3 *Variable* DEVS formalism

Here we outline an extension of DEVS formalism to represent structural changes in models. This extension is named *Variable* DEVS, V-DEVS. In V-DEVS we define atomic models as in original DEVS by:

$M \equiv <X,S,Y,\delta_{int},\delta_{ext},\lambda,ta>$

Structure changing is provided by the new *variable coupled model* defined by:

$V\text{-}CM \equiv <X,Y,\mathcal{C},select>$

where X, Y and select have the same meaning as in DEVS formalism and $\mathcal{C}$ is defined by:

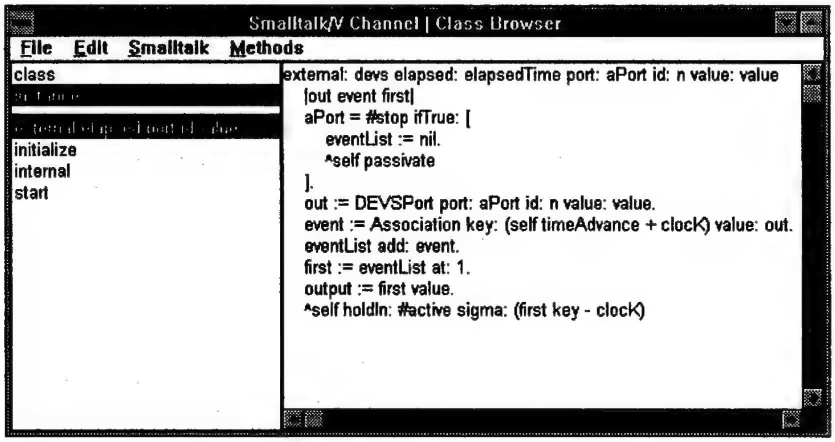
$\mathcal{C} \equiv <X,Y,\chi,\delta_{int},\delta_{ext},\lambda,ta>$

and χ is defined by:

$\chi \equiv <M,C>$

M and C represent models and connections respectively, as in DEVS.

$\mathcal{C}$ is an atomic model that handles the connections in the variable coupled model. $\mathcal{C}$ acts like the controller of the coupled model, by keeping composition and connection information. Changes in structure can be initiated only by the internal or external transitions of this element. With this definition V-DEVS keeps the modular proprieties as defined in original DEVS formalism.



```
external: devs elapsed: elapsedTime port: aPort id: n value: value
    |out event first|
    aPort = #stop ifTrue: [
        eventList := nil.
        ^self passivate
    ].
    out := DEVSPort port: aPort id: n value: value.
    event := Association key: (self timeAdvance + clocK) value: out.
    eventList add: event.
    first := eventList at: 1.
    output := first value.
    ^self holdIn: #active sigma: (first key - clocK)
```

Figure 1. Channel external function.

4 DEVS environment

DEVS formalism is implemented in the Smalltalk/V language [14]. Smalltalk is a class-based object-oriented language. To build new models and to define their proprieties we use Smalltalk/V browsers, fig. 1. In this modeling environment each model is an instance of a simulation class. The root class *DEVSEntity* is specialized in class *DEVSModel* and *DEVSProcessor*. *DEVSModel* subclasses are the actual models and *DEVSProcessor* subclasses: *RootCoOrdinator*, *CoOrdinator* and *Simulator* implement the abstract processors described in [12], and necessary to execute the model implicit behavior.

The class *AtomicModel* is the root class of all atomic model and coupled models are implemented in class *CoupledModel*. To handle changes in model structure we have created the new *VarCoupledModel* class. The co-ordinator necessary to handle the *VarCoupledModel* is implemented by the class *VarCoOrdinator*.

5 Adaptive computer architecture

One of the most promising application of variable structure modeling methodology is in modeling and simulation of adaptive computer architectures. We describe the modeling of an adaptive computer in V-DEVS formalism.

5.1 Description

In this section we briefly describe an adaptive computer architecture system and its modeling in the V-DEVS formalism previously described. Full details of this system and simulation results are presented in [15].

This computer architecture consists of a variable number of flexible computers (FCs) and a variable number of channels. FCs are connected in a tree like configuration, fig. 2.

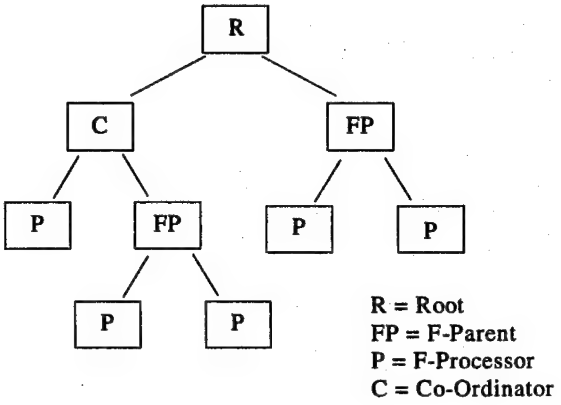


Figure 2. Computer architecture tree.

FC in the leaves of the tree (named here by f-processors) are the only processors that perform computations. Inner FCs, also called co-ordinators, redirect problems to the f-processors. This architecture is able to change its own structure to keep a desired performance. The number of FCs is increased when the

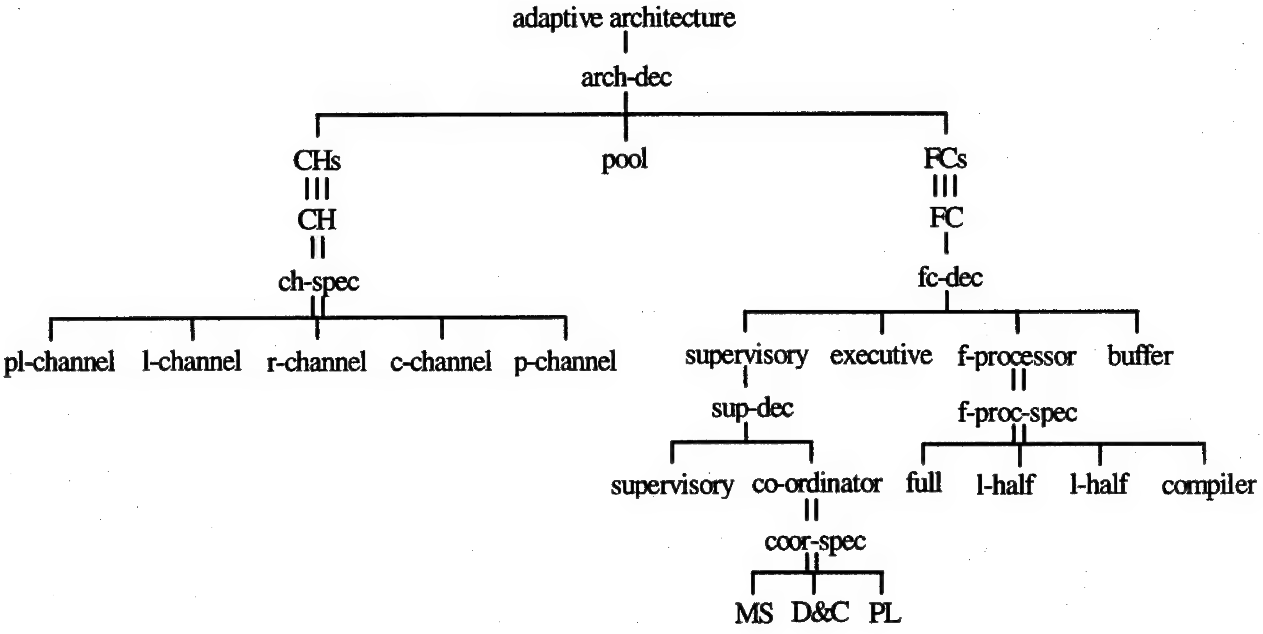


Figure 3. System entity structure for the adaptive computer architecture.

performance is low and is decreased when computer performance is above an upper limit. These operations are called hire/fire operations and change the structure of the computer architecture. FCs are connected by CHANNELS with an unbounded capacity and a constant delay.

The *System Entity Structure* (SES) provides a formalism for specifying system composition [13]. SES provides information about decomposition, coupling and taxonomy. In fig. 3 is represented the SES for the adaptive computer architecture. Instead of pruning the SES at the beginning of simulation we start with a single processor and the structure changes dynamically during simulation. SES also provides a formal framework for representing the family of possible structures.

Each FC has four modules: an EXECUTIVE, a BUFFER, a SUPERVISORY and a F-PROCESSOR. The EXECUTIVE module handles the hire/fire commands. The BUFFER stores new processes and is used in FC changes. The SUPERVISORY is used when the FC is working as a co-ordinator. The F-PROCESSOR has the task of packet (problem) processing. A simplified representation of FC connections is represented in fig. 4.

When the FC is working as a leave processor (f-processor) the BUFFER sends problems to the F-PROCESSOR module. Arriving problems are stored in the BUFFER if the F-PROCESSOR is busy. The F-PROCESSOR after solving the current problem sends a done signal asking the BUFFER a new problem.

The FC can also act as a co-ordinator. In this role arriving packetes are sent from the BUFFER to the SUPERVISORY component and from this module to one of the children FCs. The SUPERVISORY is a composition of a SUPERVISOR and a COORDINATOR. The SUPERVISOR send new problems to the CO-ORDINATOR module and solved problems to a high level processor. There are three different types of COORDINATORs: multiserver, MS (the problem is sent to the child processor with smallest queue size), divide & conquer, D&C (divided problems are sent to the child processors and partial solutions are sent to a special processor for final compilation), and PIPELINE, PL (problems pass through a chain of processors).

Architecture changes are initiated by hire/fire commands. When an f-processor, FP, receives a hire command it checks if buffer size is greater than an higher limit. In this case the FP send an hire command to the POOL and after receiving the hire confirmation it starts the hire process. In this transitory state all the incoming problems are stored in the buffer and a type of CO-ORDINATOR is chosen in the SUPERVISORY element. When the F-PROCESSOR module finishes problem processing, the buffer sends all its stored problems to the SUPERVISORY for distribution to the new f-processors. After the hire operation the f-processor becomes an f-parent.

The fire operation can be performed only by f-parents (co-ordinators that all its children are f-processors). When an f-parent receives a fire command it will check buffer size; if buffer size is lower than a minimum limit the f-parent begins the fire process. Incoming problems are stored in the buffer, and the FP waits until all problems still in the children are solved and sent back to the CO-ORDINATOR. When all children are empty the CO-ORDINATOR informs the executive that it can proceed with fire operation. The EXECUTIVE releases

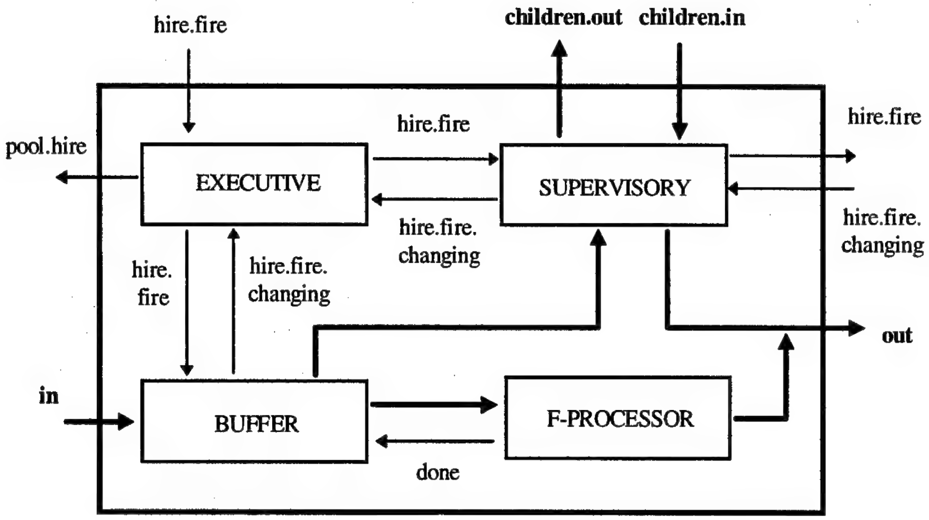


Figure 4. Simplified representation of F-COMPUTER internal coupling.

FCs children in the POOL and informs the BUFFER to start acting in f-processor operation mode. The BUFFER sends now a problem to the F-PROCESSOR module and the FC starts the operation as an f-processor.

In order to correctly implement hire/fire policies each EXECUTIVE must have knowledge of its position in the computer architecture tree. The current implementation follows the principles pointed in [13], [16] for knowledge representation.

Channels provides an unidirectional communication links between processors themselves and between processors and the POOL. Each channel is modeled with a fixed delay and no transmission limits. A p-channel connects an FC with its parent, a l/r-channel a co-ordinator with its right/left processor, a c-channel connects a D&C co-ordinator with the compiler processor, and the pl-channel connects an FC with the POOL.

5.2 Structure change decisions

For supporting changes in structure is necessary to decide where these changes are generated, how they are propagated and accepted, and which co-ordinator to be selected. The hire/fire generation policy determines where changes commands have their origin. The actual model supports a centralized policy and a distributed policy for hire/fire generation. In the distributed policy hire commands are generated in the leaves f-processor and fire commands are generated by f-parents. In a centralized generation policy both hire/fire commands are generated by the Root processor.

The transmission policy depends on the kind of co-ordinator. The MS co-ordinator passes hire/fire signals to the child with the longest/shortest queue length. The D&C and PL co-ordinators send hire/fire commands to both r-child and l-child if the number of problems is greater/lower than a the hire/fire limit.

The accept policy decides about actually hire/fire command execution by the lower level processors. An f-parent decides to accept a fire command if the number of problems is lower than the fire limit. It sends a fire message to the POOL and it becomes an f-processor after its children become idle. In fig. 5 is depicted the changes in the structure provoked by fire command sent by the root processor and accepted by an f-parent.

The hire command is accepted by an f-processor if the number f problems packets is higher than the hire limit. Upon hire acceptation the f-processor sends a hire command to the POOL. If there are enough processors in the pool (2 FCs for MS and PL co-ordinators, and 3 FCs for D&C co-ordinator) it becomes an f-parent.

5.3 Implementation

The POOL element is the key for structure change. This element starts change structure commands in response to its external function. After receiving a hire command the POOL checks the number of available processors. If there are enough free processors the POOL connects new processors to the hiring f-processor. The POOL also creates CHANNELS that models the physical links that exists between processors. Changes in POOL internal structure are automatically updated in the model. When the POOL receives the fire command it removes leave processors and increases the number of free processors.

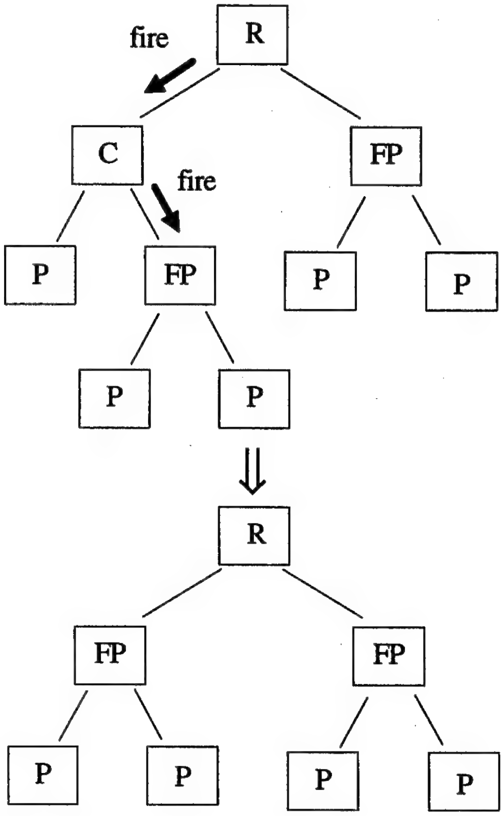


Figure 5. Computer architecture after a fire command.

In fig. 6 is represented an abbreviated version of the external transition of POOL. The depicted transition handles hire/fire commands for MS and PL co-ordinators. For the D&C co-ordinator we have to hire/fire 3 processors instead of 2 required in the other co-ordinators.

```
external: devs elapsed: elapsedTime port: aPort id: n value:
value
        ...
  aPort = hire ifTrue: [
        ...
    left := DEVSEntity fcomputer: (name,'@Left')
      random: randomGenerator
      bounds: minMax
      policy: hireFirePolicy
      root: false hireFire: time.
    lcPool := Channel pair: (name,'@LCPool')
      random: randomGenerator.
    lcDown := Channel pair: (name,'@LCDown')
      random: randomGenerator.
    lcUp := Channel pair: (name,'@LCUp')
      random: randomGenerator.
        ....
    self addChild: left. self addChild: lcDown.
    self addChild: lcUp. self addChild: lcPool.
        ...
    depend := OrderedCollection
      with: left with: lcPool
      with: lcDown with: lcUp.
        ...
    tree at: fcomp put: depend.
        ....
        "Left Down Channel"
    self couple: fcomp port: #out to: lcDown port: #out.
    self couple: fcomp port: #hf to: lcDown port: #hf.
    self couple: lcDown port: #out to: left port: #IN.
    self couple: lcDown port: #hf to: left port: #hf.
        ....
  ].
  aPort = #fire ifTrue: [
        ...
    uProcessors := uProcessors + 2.
    dup := (tree at: value) copy.
    dup do: [:x| self removeChild: x].
    tree removeKey: value.
    ^self continue: elapsedTime
  ].
        ...
  ^self error: 'Unknown Port ',aPort
```

Figure 6. Simplified code for Pool, MS and PL, hire/fire operations.

The method **addChild:** is used to add new models to the simulation. To remove models we use the **removeChild:**. Connections are established with the method **couple:port:to:port:**. When a model is removed its connections are also removed.

6 Conclusions

We described *Variable* DEVS an extension of DEVS formalism that is able to represent structural changes in simulation models. This formalism keeps modularity as defined in original DEVS. V-DEVS proved to be able of modeling a complex adaptive computer architecture. As future work we plan to incorporate SES with our variable structure environment for automatic structure change.

References

[1] Zeigler, B.P. and R.G. Reynolds. 1985. "Towards a Theory of Adaptive Computer Architectures". *Proc. 5th International Conference on Distributed Computing Systems*. Computer Society Press, 468-475.

[2] Zeigler, B.P. 1986. "Toward a Simulation Methodology for Variable Structure Modelling". In *Modelling and Simulation Methodology in the Artificial Intelligence Era*. M.S. Elzas, T.I Ören and B.P. Zeigler, eds. North-Holland, 195-210.

[3] Zeigler, B.P. 1989. "Concepts for Distributed Knowledge Maintenance in Variable Structure Models". In *Modelling and Simulation Methodology in the Artificial Intelligence Era*. M.S. Elzas, T.I Ören and B.P. Zeigler, eds. North-Holland, 45-54.

[4] Kim, J.W. 1994. *Hierarchical Asynchronous Genetic Algorithms for Parallel/Distributed Simulation-Based Simulation*. Ph.D. Dissertation. Department of Electrical and Computer Engineering. University of Arizona.

[5] Zeigler, B.P. and A. Louri. 1993. "A Simulation Environment for Intelligent Machine Architecture". *Journal of Parallel and Distributed Computing* 18: 77-88.

[6] Chean, M. and L.A.B. Fortes. 1990. "A Toxonomy of Reconfigurable Techniques for Fault-Tolerant Processor Arrays". *IEEE Computer* 23, no. 1 (Dec.): 55-69.

[7] Uhrmacher, A.M. 1993. "Variable Structure Models: Autonomy and Control - Answers from Two Different Modeling Approaches". *Proc. AI, Simulation, and Planning in High Autonomy Systems*. IEEE Computer Society Press, 133-139.

[8] Zeigler, B.P. and H. Praehofer. 1989. "Systems Theory Challenges in the Simulation of Variable Structure and Intelligent Systems". In *CAST-Computer-Aided Systems Theory*. F. Pichler and F. Moreno-Diaz, eds. Springer Verlag, 41-51.

[9] Vasconcelos M.J.P., J.M.C. Pereira and B.P. Zeigler. 1993. "Simulation of Fire Growth in GIS Using Discrete Event Hierarchical Modular Models". *Proc. Satellite technology and GIS for Mediterranean Forest Mapping and Fire Management.*

[10] Uhrmacher, A.M. and B.P. Zeigler. 1994. "Variable Structure Models in Object-Oriented Simulation". *International Journal of General Systems*. (Accepted for publication.)

[11] Cho, T.H. 1993. *Hierarchical Modular Simulation Environment for Flexible Manufacturing System Modeling*. Ph.D. Dissertation. Department of Electrical and Computer Engineering. University of Arizona.

[12] Zeigler, B.P. 1984. *Multifaceted Modelling and Discrete Event Simulation*. Academic Press.

[13] Zeigler, B.P. 1990. *Object-Oriented Simulation with Hierarchical, Modular Models: Intelligent Agents and Endomorphic Systems*. Academic Press.

[14] Barros, F.J. and M.T. Mendes. 1993. "Modelling and Simulation of an Integrated Circuit Assembly Flow-Shop in DEVS-V". *Proceedings of the European Simulation Symposium*. SCS publications, 103-108.

[15] Wang, I.Y. 1986. *Simulation of a Modular Hierarchical Adaptive Computer Architecture with Communication Delay*. MS Thesis, Department of Electrical and Computer Engineering. University of Arizona.

[16] Zeigler, B.P., T.G. Kim and C. Lee. 1991. "Variable Structure Modelling Methodology: An Adaptive Computer Architecture Example". *Transactions of The Society for Computer Simulation* 7, no. 4: 291-319.

Verb Phrase Model Specification via System Entity Structures

Richard J. Simard, Rome Laboratory (USAF)
Bernard P. Zeigler, University of Arizona
Jerry M. Couretas, University of Arizona

Abstract

In investigating front end model development, an environment is described that allows for model construction through pruning a domain specific System Entity Structure. The preformal stages of the model will be represented by a verb phrase. This representation is sufficiently detailed to serve as the basis for model construction and yet sufficiently "soft" to support knowledge acquisition during model construction. This paper establishes the adequacy of this representation.

Introduction

There are problems in the current modeling process associated with model development. Although the "back-end" of the model-engineering process Fishwick [1] is well-supported by software tools, the same can hardly be said for the "front-end" -- model creation, construction, or repository-based synthesis.

This paper suggests that such "front-end" problems can be substantially ameliorated by adopting techniques that allow the user to narrow down essential components for model construction. The particular objectives for model construction that we wish to address are extracted through a natural language interface.

The goal of this approach is to reduce ambiguity between the user's requirements and essential model construction components. Requirements here are represented by verb phrases. Each one has the potential to be incorporated into a complete discrete event model. This natural language process describes a method of constructing models by users with a limited or nonexistent formal modeling or programming background.

Overview

Paradigms for transforming the meaning of sentences into conceptual modeling structures have been proposed by Heidorn [2], Howard W. Beck and Paul A. Fishwick [3]. The intent of this research is directed toward developing a system for performing simulation analysis through natural language interaction with a computer. This work represents a "front-end" -- model creation, construction, or repository-based synthesis process.

A natural language interface allows model specification in terms of a verb phrase. It consists of a verb, noun, and modifier. An example might be "build car quickly." In this case the verb is build, the noun is car, and the modifier is quickly. The verb "build" would be parameterized by the noun "car." The noun specifies the domain that the verb interacts with. The term "parameterized by the noun" is used in our context to mean the descriptive components (parameters) of the noun that provide further detail about it's domain. For example, in "build car quickly," the car would have parameters such as length, weight, etc. The modifier "quickly" adds range to the parameterization. This is similar to Zadeh's [4] use of fuzzy restrictions. An example verb phrase is shown in Figure 1.

Conceptual realization of a model from a verb phrase ties in closely with Checkland's [5] idea of having a verb express the root definition, or core purpose, of a system. He explains "That core purpose is always expressed as a transformation process in which some entity, the 'input', is changed, or transformed, into some new form of that same entity, the 'output'." Using the verb as a discrete event model template is an extrapolation of this idea.

With the basic discrete event model defined from the root definition, or verb, the noun of the verb phrase gives the user a working domain. The verb phrase's noun defines the domain of possible action. For example, "build car" parameterizes the domain in units of the mean time to build a car. This might be hours for a car. When applied to a house - "build house," this might be months.

The modifier adds focus to the noun's domain. For example, in "build car quickly," the modifier "quickly" optimizes all car building parameters. The effect of the modifier could be quantified at the user's discretion.

0-8186-6440-1/94 $04.00 © 1994 IEEE

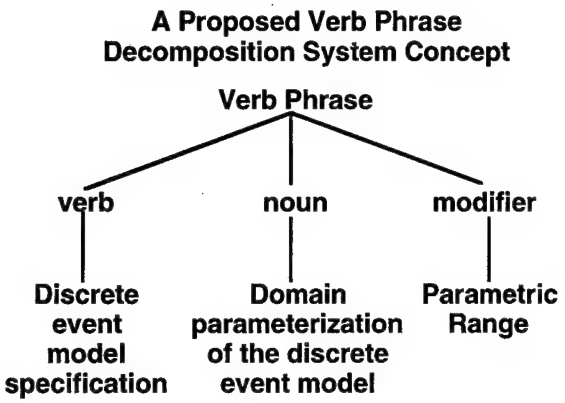

A Proposed Verb Phrase
Decomposition System Concept
Verb Phrase
verb
noun
modifier
Discrete event model specification
Domain parameterization of the discrete event model
Parametric Range

Figure 1. Verb Phrase Decomposition.

With each model represented by a verb phrase, a system could be decomposed into a semantic representation as shown in Figure 2. In this case, a hierarchical car production system goes from production planning to practical assembly line issues - with each step represented by a verb phrase. The focus is not so much on the system or its structure, but on how the natural language interface allows semantic representation of the system.

This approach focuses on description of the front end model construction process. Each system function would be modeled as a verb phrase. The verb represents the basic action to be taken, and forms the basis of the discrete event model. A noun then defines the domain. The modifier defines how the action will be performed within the domain. The verb-noun-modifier triplet defines the verb phrase. Now we will look at this definition in more detail.

Description of verb phrase extraction process

The following is a complete description of how the verb phrase is constructed. Examples of verb definition, noun parameterization, and modifier parameters of limitation are given.

I. Verb description

Using Fernald's [6] definition, the verb is transitive in requiring the verb phrase's noun to complete its meaning. It is principal in expressing the act to be done. And, its voice is either active or passive because the subject can either be acting or acted upon. Using Fernald's definition, we can develop a description of verb use in this system.

Verb definition

Verbs are used to describe the two main categories of system behavior, which we propose to be production and consumption. These verbs form the basis for each model. We also propose the verb to represent the basic discrete event model template.

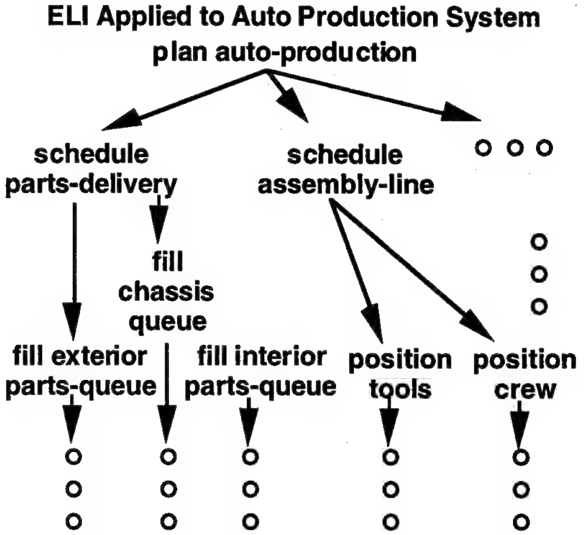


Figure 2. ELI representation of car production system hierarchy.

The first class of verbs, which we call producers, describe value added activity. This could be anything from growing crops to stamping sheet metal into automobile body panels. A producer is active. It models something that acts on the noun of the verb phrase.

Consumption is less straightforward. Consumption can be sensory or physical. Sensory consumption is defined as what we consume or observe. An example of sensory consumption could be the use of the five senses in observing one's environment. Physical consumption might be taking in nutrients for processing into energy.

Verb definition is performed by descending a verb hierarchy until a satisfactory representation is found. For example, if one's goal is building, we descend the verb's hierarchy until choosing "build" under the producer class, as shown in Figure 3.

Verbs, regardless of their class, are tied to reality by their domain. For example, the verb "build" can be used to build a car with its variables describing the action. In this case of build, the variables might be cycle time, feed rate, idle time, etc. The user controls the modification of these variables.

The variables of a verb might be in the form of an array. For example, the verb "build" might have the following variables:

build = s[feed-rate][cycle-time][idle-time]

In this case, the variables of "build" are: feed-rate, cycle-time, and dle-time. And the numerical parameter ranges would be filled in when the noun, and its accompanying domain, is chosen.

We are proposing that each verb is represented by a discrete event model shell with state variables that can be used to describe its behavior. These variables would be open for user modification. In order to simplify the understanding of verb use in this system, we also propose that they fit into a taxonomy.

Verb Definition

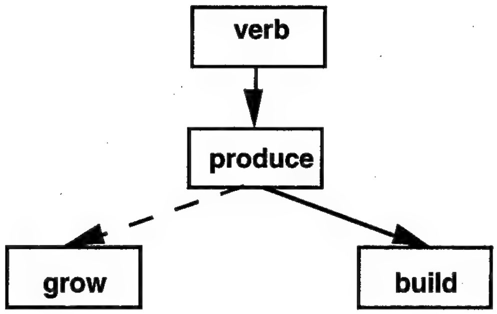


Figure 3. Producer verb <u>build</u> is chosen.

Verb taxonomy

This breakdown of verb classes gives us structure in thinking about verbs and the actions that they represent. As shown in Figure 4, the actions of producers and consumers are different. We believe this breakdown provides the user added structure when describing system requirements for model construction.

Verb Taxonomy

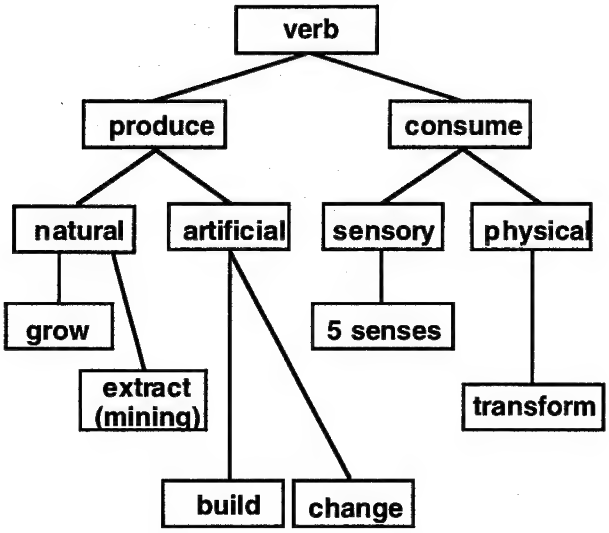


Figure 4. Taxonomic breakdown of verb into classes of production and consumption.

Verb classification

The key to verb classification is what transforms it and how. Some potential verb representing transformations are shown in Figure 5.

In looking at Figure 5, we see the following verb transformations:

Producers - Value is added to the initial state of the noun. The end state is a transformation from scrap metal to a useful automobile.

Consumers - The final state of the noun is transformed from its initial state of an automobile to scrap metal.

State Change Transformation of Producers and Consumers

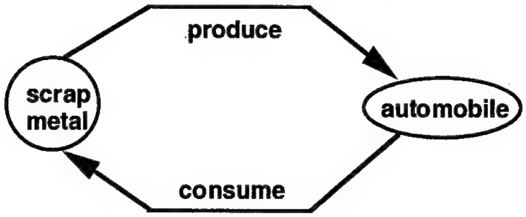


Figure 5. Verb Classification State Changes.

Producers and consumers could work together within a system. This classification system gives the user a simple way of classifying almost any process in a system. In the car production example of Figure 2, an automotive assembly line consumes component parts and produces complete automobiles. We know that we are dealing with a relatively high level system when it has both classes of verb in its definition. Decomposing this system, as shown in Figure 6, shows how the different components act as consumers and producers.

Looking at the above verb classification, we see that the main difference between producers and consumers is their relationship to the noun. A producer acts on the noun and a consumer is acted upon by the noun.

Produce	**Consume**
material > finished product	sensory data > information
location(1) > location (2)	material > energy
information > report	land > contaminated soil
layman > professional	young > old (physical capability)
undeveloped land > building	current > obsolete

Suh [7] has a comparable method where each functional requirement maps into a set of data parameters. Functional requirements account for the functional space of the design, while the data parameters account for the physical space. The mapping between these two is determined by the design, or verb phrase system representation. A natural language interface is similar in that each verb represents a functional requirement. And, the verb's domain is defined by data parameters. Verb name specification would result in the system accessing a database of verbs along with their parameterization as shown below:

verb = < param1, param2, ... >
build = < feed-rate, cycle-time, queue-time, ... >

In this case, the verb "build" is parameterized by feed-rate, cycle-time, queue-time, etc. This would be the knowledge based component of the system. Whenever a verb is called, it would be parameterized as previously defined. The user would then be free to modify the parameters given.

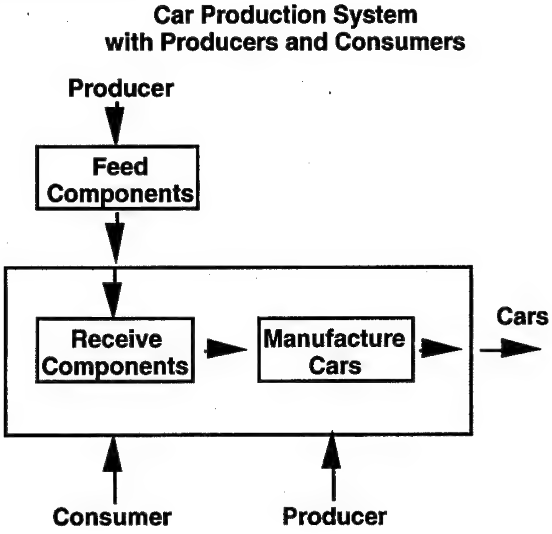


Figure 6. Example system with different model types.

The verb represents one of the system's actions. It also represents a basic discrete event model with an empty set of descriptive variables. These are open for user modification, and would be parameterized by the noun.

II. Noun description

The noun is the second component for construction of the verb phrase. The noun serves as the object of the verb phrase and thereby establishes its domain parameterization. The noun's definition and taxonomy are presented next.

Noun definition

The noun defines the domain in which the verb works. Each noun has a different domain representation, and thus has a different parameterization. The noun's domain knowledge should provide the parameterization necessary to complete the model. The noun quantifies the verb's action parameters. For example, the verb "build" could be parameterized as follows when applied to the noun "car":

build = s[feed-rate] [cycle-time] [idle-time]
build = s[5] [3] [1]

This means that the verb phrase "build car" breaks down to the following numerical parameterization:

feed-rate	=	5
cycle-time	=	3
idle-time	=	1

The noun's domain will define the numerical parameters of the verb's model. Another view of the noun is presented by its taxonomy.

Noun taxonomy

The noun decomposes into objects. These can be classed according to their environment and key parameters. One decomposition of the nouns might be physical, mental, man made objects, and natural objects. Many of the practical things we deal with, such as an automobile, are found by descending this hierarchy.

In modelling people, both mental and physical characteristics are of interest. Physical capacity, or how much force the body can be subjected to, might help someone designing an ejector seat, or allow checking the amount of force a person experiences during different maneuvers. Physical strength might be of interest in checking the force required to turn a knob or open a door. Thus, a person's physical modeling could be of either capacity or strength.

Mental capabilities are also of interest to the modeller. Certain tasks of abstraction may require modeling to get the right personnel fit. Also, mental endurance could be of interest for challenges like operating a vehicle for long periods of time or withstanding certain environments. Capacity and endurance are representative of these types of mental modeling.

Man made structures could be anything people make. Natural structures include things ranging from the tiniest crystal on a snowflake to our conception of the universe.

The noun, as shown in Figure 7, decomposes into the general classes of people and things.

The noun provides focus needed for parameterization of the verb. With the verb-noun pair, the user has a complete verb phrase. Further definition to the noun's parameters could be done with the use of modifiers. Modifiers are used to change the verb phrase's parameterization to better fit the user's conception of the process he wishes to model.

III. Modifier

Modifiers limit the parametric range of the verb phrase. Similar to Zadeh's [4] fuzzy restrictions, they help the user construct a better representation of what he means. Modification also impacts the model's numerical parameterization.

Noun Taxonomy

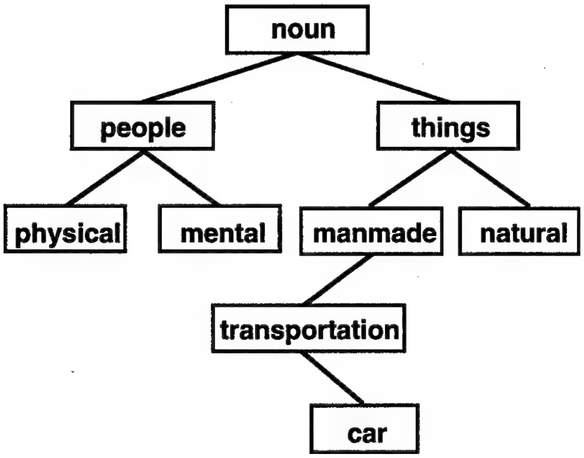


Figure 7. Noun representation in different possible classifications.

Modifier definition

The verb phrase's modifier further defines the verb phrase. Modifiers are placed directly after the noun as follows:

verb noun mod(n) mod(n-1) ... mod(1)

The user will choose the modifier(s) when defining the verb phrase representation of the model. Each modifier adds a level of degree to the action performed. For example,

build	**car**	**quickly**
verb	noun	modifier(1)

could modify the verb build by increasing the feed rate, reducing the cycle time, and reducing the idle time. Adding another modifier,

build	**car**	**very**	**quickly**
verb	noun	modifier(2)	modifier(1)

would result in a further reduction in build parameters, possibly to their lower limit.

At this point, extension would continue according to the user's discretion.

Modifier methods

Verb phrase modification comes through increased narrowing of domain parameters. One way to present this idea would be to make each modifier a multiplier of the noun parameters that it will affect. For example, "build car quickly" could be numerically broken down as follows:

Parameters	build car		quickly		build car quickly
feed-rate	5		0.6		3.0
cycle-time	3	X	0.6	=	1.8
idle-time	1		0.6		0.6

In this example, "quickly" translates to 60% of the normal build time. We are assuming that this is a simple system where reducing each of the incoming parameters to 60% of their original value results in an equivalent reduction in build time.

Another layer of modification might be "build car very quickly." "Very" serves as an additional parameter modification as shown below:

Parameters	build car		very		quickly		build car very quickly
feed-rate	5		0.5		0.6		1.5
cycle-time	3	X	0.5	X	0.6	=	0.9
idle-time	1		0.5		0.6		0.3

The modification process could be continued on until the minimum of each parameter limitation is reached.

This is a very simple example of the modifier's ability to affect the verb phrase parameters. Extensions could be done in how the parameters are modified, when parameter modifications occur, etc.

Modifiers affect parameters in correlation with their semantic meaning. Modifier interpretation would be up to the user. How much is "very?" or, how fast is "quickly?" In time, research might show that these modifiers are generally quantifiable over a range of domains. But, for now, this will be up to the user.

Verb phase extraction from SES

With the method of constructing a verb phrase clear, we can now move on to how model construction would actually occur. The process basically consists of pruning a domain specific System Entity Structure - Zeigler [8,9]. This stores all models pertinent to a specific domain as shown in Figure 8.

Once the domain is specified, pruning is a matter of narrowing down the best representation of each element of the verb phrase. For example, in narrowing down the verb, descending the hierarchy to pick the class of the verb would be done first.

Further definition results in the pruning of the exact verb to be used. Shown in Figure 9, "build" is chosen from possibilities, and is moved up the hierarchy to the verb node.

Similarly, the noun is first pruned in terms of its class, "thing," and then down to the actual entity. In the same way, the modifier is extracted by first deciding on the modifier class, "concentration," and then down to the actual modifier. This is shown in Figures 10 and 11.

System Entity Structure of Car Manufacturing Domain

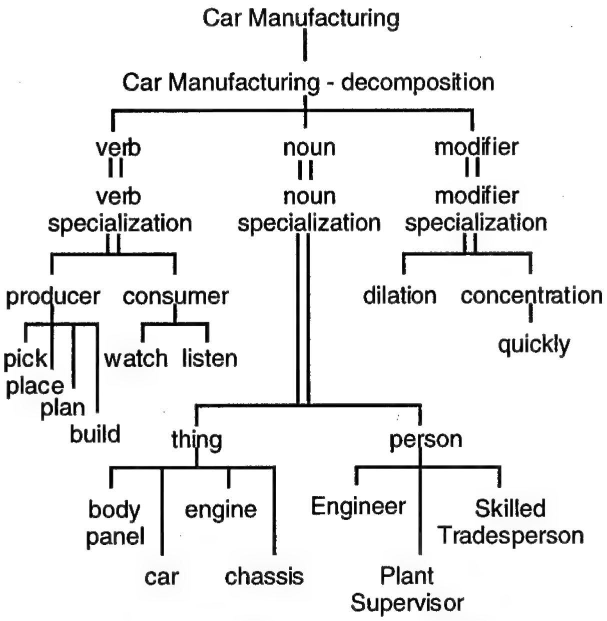


Figure 8. Simplified domain representation for car manufacturing.

Verb Pruning in Car Manufacturing Domain SES

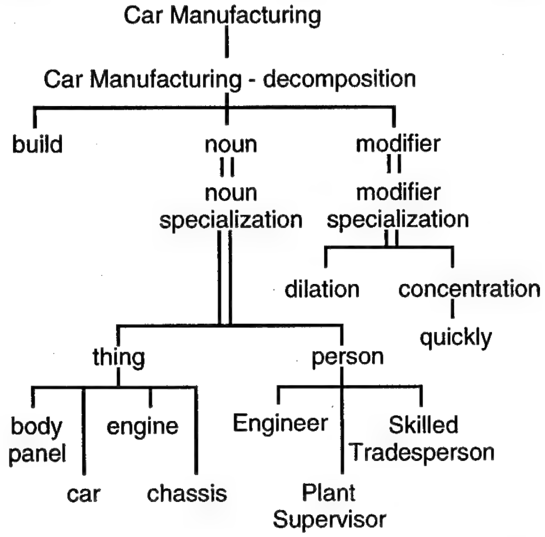


Figure 9. Prune verb "build" from producer class and replace the verb node with it.

Generating simulation models

With the verb phase pruned, the next step is pruning the model base of entities that can actually carry out what the verb phase specifies. This pruning is a two step process. First, the SES for "build" is accessed. Next, using Rozenblit and Huang's [10] Frames and Rules Associated System Entity Structure (FRASES), prune the exact model specified by the verb phase.

Noun Pruning in Car Manufacturing Domain SES

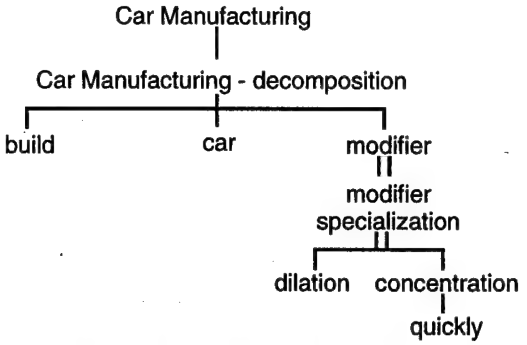


Figure 10. Noun is pruned down to the "car" to be operated on by the verb "build".

Modifier Pruning in Car Manufacturing Domain SES

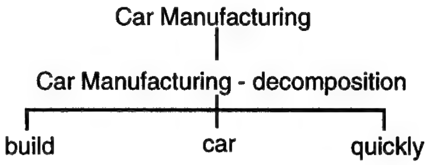


Figure 11. Class specialization "concentration" is replaced by modifier "quickly" in completing the verb phrase

The "build" SES would be in the following form as shown in Figure 12.

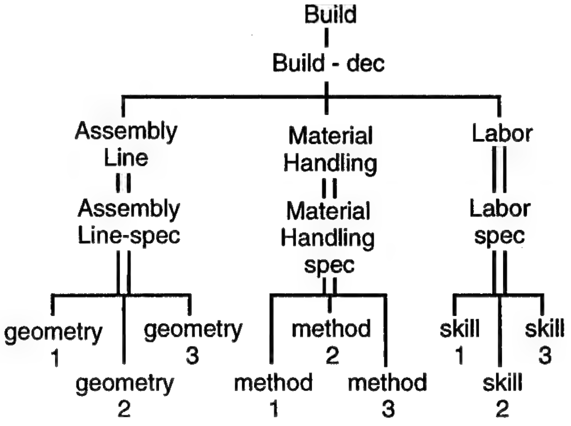


Figure 12. SES of Build Domain.

Using the FRASES methodology, rules exist at each context sensitive node. Rules for the assembly line node might be:

R1. if noun a car
then assembly line is geometry 1
R2. if noun is truck or mini-van,
then assembly line is geometry 2
R3. if noun is commercial truck
then assembly line is geometry 3

When the assembly line geometry is a scaling dependent on the type of vehicle being worked on, Geometry 1, the smallest is used for cars. And geometry 3 would be the largest; used for commercial trucks.

Likewise, example labor rules might be:

R1. if modifier is slowly
    then skill level is skill 1
R2. if modifier is quickly
    then skill level is skill 2
R3. if modifier is very quickly
    then skill level is skill 3

And, material handling rules could be:

R1. if modifier is slowly
    then coordination method is method 1
R2. if modifier is quickly
    then coordination method is method 2
R3. if modifier is very quickly
    then coordination method is method 3

A pruned model structure for "build car quickly" then looks like figure 13.

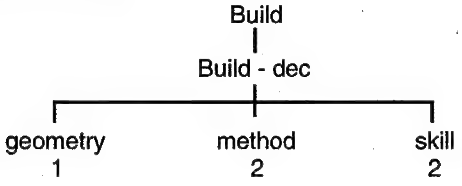


Figure 13. Pruned SES for "Build Car Quickly".

And "build truck slowly" looks like figure 14.

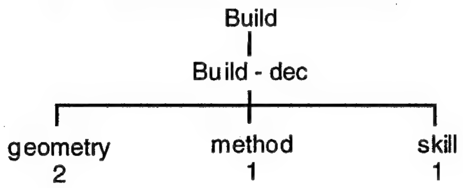


Figure 14. Pruned SES for "Build Truck Slowly".

Conclusion

The system proposed here allows for complete model description in terms of a verb phrase. The verb, representing the action to be performed, accesses a discrete event model primitive. The verb's state variables would then be numerically parameterized by the noun's domain. Further control of the domain comes through the modifiers. These are placed after the verb-noun pair and modify the entire verb phrase. The verb phrase consists of a verb-noun-modifier(s) triplet that represents a discrete event model.

This method considers four layers of System Entity Structure in creating a discrete event model from an English language verb phrase:

1. Sentence clarification SES
2. Domain identification SES
3. Model representation SES
4. Model storage SES

The above shows the inherent organizing power of an SES. Four level of knowledge representations are proposed to transform a verb phrase into a discrete event model.

This representation is not complete. There are quite a few possibilities for expansion of this system. The verb phrase could be expanded to add clarification. Verb phrases could be combined for multiple action. And, the practical issues of dealing with parameters have not been addressed. The verb phrase described here is simply a way to give the user an intuitive model representation.

Using a verb phrase for model representation could open the domain of modeling to a larger group of users. We believe that the main barrier between many people and existing modeling software is their lack of computer literacy. They might use a natural language interface as a means of bridging this gap. Verb phrase representation could create modellers out of people who think semantically, and have no computer skills.

A semantic representation frees the user to explore the system on the familiar grounds of natural language. A verb phrase system might open the way for brain storming, innovation and proving out of ideas before they leave the drawing board. In addition, more ideas could be checked and compared before deciding on the best option. The potential applications that might result from expanded use of discrete event modeling are wide open. And a natural language interface could lead the way.

References

1. Fishwick, P.A., (1990), "Toward an Integrated Approach to Simulation Model Engineering," Int. J General Systems, Vol. 17, pp. 1-19.

2. Heidorn, G.E., (1972), Natural Language Inputs to a Simulation Programming Language, Ph.D. Dissertation, Yale University.

3. Fishwick, P.A. and Beck, H.W., (1989), "Incorporating natural language descriptions into modeling and simulation" Simulation, Simulation Councils, Inc. Vol. 52:3, pp. 102-109.

4. Zadeh, L.A. (1975), Calculus of Fuzzy Restrictions, Academic Press, New York, NY.

5. Checkland, Peter, and Scholes, Jim, (1990), Soft Systems Methodology in Action, John Wiley & Sons, New York, NY.

6. Fernald, James C., (1957), English Grammar Simplified, Funk & Wagnalls Company, New York, NY.

7. Suh, Nam P. (1990), "The Principles of Design," Oxford University Press, Inc., New York, NY.

8. Zeigler, Bernard P. (1990), "Intelligent Agents and Endomorphic Systems," Academic Press, London.

9. Zeigler, Bernard P. (1984), "Multifacetted Modelling and Discrete Event Simulation," Academic Press, London.

10. Rozenblit, J. W. and Y. Huang (1987), "Constraint-Driven Generation of Model Structures", Proc. of Winter Simulation Conf., SCS Publications, San Diego, CA.

A Framework for Hybrid Modeling/Simulation of Discrete Event Systems

Myung Soo Ahn and Tag Gon Kim

Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
373-1 Kusong-Dong,Yusong-Gu,Taejon 305-701, Korea

Abstract

This paper presents a hybrid modeling/simulation framework within which both accuracy in models and speed in simulation experimentations are obtained. Based on the Zeigler's DEVS formalism and associated system theory, the framework is based on the transformation of selected DEVS models into equivalent analytic ones to simulate both analytic and simulation models within a single environment. For high-speed hybrid simulation, we extended DEVSim++ which is a realization of the DEVS formalism in C++. To exemplify the proposed approach, we demonstrate performance modeling and simulation of a simple communication network.

1 Introduction

Recently discrete event modeling and simulation for performance evaluation of complex systems becomes an important research issue in many areas of system design such as manufacturing systems design, communication networks design, and realtime systems design. The main research objective is to devise a framework for developing accurate performance models and efficient simulation algorithms for fast experimentations with such models[3, 5]. Such a framework is essential to reduce the computation burden for simulating large complex systems.

Two types of models, analytic and simulation, have been applied in system analysis and performance evaluation. Although analytic models have limitations in accuracy due to the unrealistic assumptions made, fast simulation experimentations are possible by employing appropriate numerical analysis technique. On the other hand, simulation models have expressive means powerful enough to achieve high accuracy. However, simulation time for such models is extremely slow compared with that for analytic models due to some virtual management scheme embedded in a simulation algorithm. Thus, it is highly desirable to combine the advantages from both models in an unified framework[5].

Up to date, little research has been reported concerning modeling and simulation methodologies which meet both accuracy in modeling and speed in simulation within an unified framework. Furthermore, no simulation language or environment implementing such methodologies is in place.

The purpose of this paper is to develop a framework within which both accuracy in models and speed in simulation experimentations are obtained. To be specific, for modeling we propose a model transformation scheme which transforms selected simulation models into analytic ones as far as accuracy is preserved. For simulation, we develop a hybrid simulation algorithm and the associated environment implementing such algorithm. Thus, we can simulate performance models of a discrete event system, which consists of simulation models and analytic ones, in a single environment.

We employ the Zeigler's DEVS formalism[8], which supports hierarchical modular descriptions of discrete event systems. Though the set-theoretic formalism has expressive power and the well known simulator algorithm, it lacks of analytic means. To complement

[0] ISBN 0-8186-6440-1. Copyright ©1994 IEEE. All rights reserved.

0-8186-6440-1/94 $04.00 © 1994 IEEE

this shortage, the model transformation scheme transforms a DEVS model into a behaviorally equivalent analytic model in steady state. Both DEVS models and transformed analytic models are simulated in a combined manner using the developed hybrid simulation engine.

Our approach is novel in the sense that :

- it employs a single modeling formalism. Thus, it enables the modelers to develop models within the expressive formalism.
- it is based on the sound mathematical foundations in the behavioral equivalence relation between the DEVS models and the transformed analytic models.

The developed framework can be used to measure the performance of a discrete event system with greatly reduced simulation time. To exemplify the proposed hybrid modeling and simulation framework, we demonstrate performance modeling and simulation of a simple communication network.

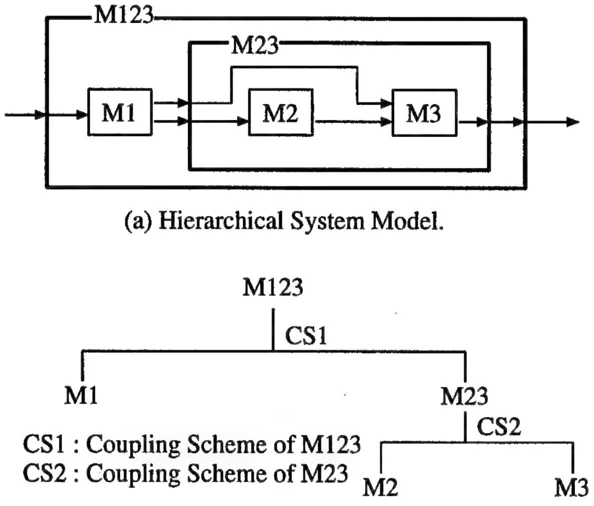
The outline of this paper is as follows. Section 2 presents the concepts of hybrid modeling. Section 3 reviews the DEVS formalism and describes our framework. Section 4 gives an example of application and results. We conclude the paper in section 5.

2 Hybrid Modeling with Hierarchical Discrete Event Models

Hierarchical constructions of modular models play an important role in modeling and simulation for design of complex real world systems. In hierarchical composition, higher-level models include low-level models as components. Such higher-level models can be reused as components of yet higher-level models[6].

In such a construction, an atomic model is a basic system component that can function as a self-contained and independent unit. Such a model interacts with other models only through the input and output interface, thereby achieving modularity. Thus, an atomic model may well be characterized by a system that can be defined by an input/output interface and state transitions, which represent the behavior of the model.

Systems may be coupled to build coupled models, which may themselves be employed as components to be coupled with other models to form higher level systems. A coupled model specifies the coupling scheme, that is, input/output connections between component models. A simulation model developed by the hierarchical composition methodology has the structure of a tree, called decomposition tree. Figure 1 shows a hierarchical modeling of a system and associated decomposition tree.

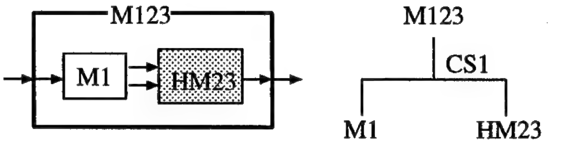


(a) Hierarchical System Model.

(b) Decomposition Tree.

Figure 1: Hierarchical Modular Model.

When such a hierarchical simulation model is given, it is possible to transform some or all of component models into the equivalent analytic models. We call this transforming process as hybrid modeling. The hybrid modeling proceeds in bottom-up fashion. That is, it first transforms the atomic models that meet transformable conditions. After transforming all the possible atomic models, it then aggregates the transformed analytic models using the coupling information. Figure 2 shows an hybrid model which is the result of transformation and aggregation of M12 in Figure 1.



(a) Hybrid Model. (b) Decomposition Tree.

Figure 2: Hybrid Modeling of Hierarchical Model.

Most analytic models require some constraints to make mathematically tractable ones. One of such constraints is the Markov property in continuous Markov chain model. Thus, we need assumptions or simplification methods for transforming atomic simulation models. Two methods can be considered :

- Simplification : state space reduction
  - discretization of continuous state variables;
  - grouping of states at the activity level;
- Generalization : steady state assumption
  - Poisson process assumptions for all event types;
  - Exponential distributions of service times.

But, not all atomic models are transformable. Especially, models representing priority-based processors can not be transformed. Hybrid modeling does not transform such models, thereby preserving accuracy of models. This is the main advantage of hybrid modeling.

The aggregation step at the coupled model level merges two or more transformed analytic models in a single model by using the coupling scheme. Consider two models M1 and M2, shown in Figure 3. M1 acts as a buffer and M2 is a processor. Generally, an atomic model falls in one of these two classes of models : buffer or processor. Using the coupling scheme of two models, we can construct a simple queueing model by merging M1 and M2.

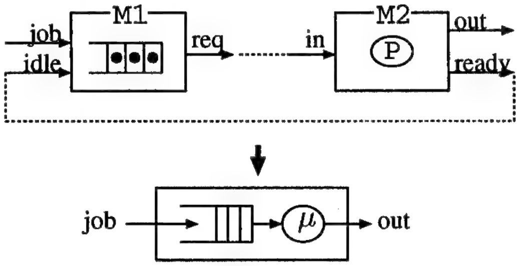


Figure 3: Merging of Two Models into a Single Model.

We repeatedly apply the aggregation process between components models. If all component models are merged into a single model, then the coupled model is transformed in an analytic model. Other coupling information can be useful for merging the models. As an example, a tandem connection of queues can be represented as a single queue[4]. After transforming all the possible models, the hybrid model is simulated in a combined manner.

3 Hybrid Simulation within the DEVSim++ Framework

For hybrid modeling and simulation, we employ the Zeigler's DEVS formalism[8]. As shown in Figure 4, we extend DEVSim++[2] by adding two schemes. The first scheme is a model transformer which transforms selected DEVS models into the equivalent analytic models. The second one is a hybrid simulation engine. This section briefly reviews the DEVS formalism and explains our hybrid simulation environment.

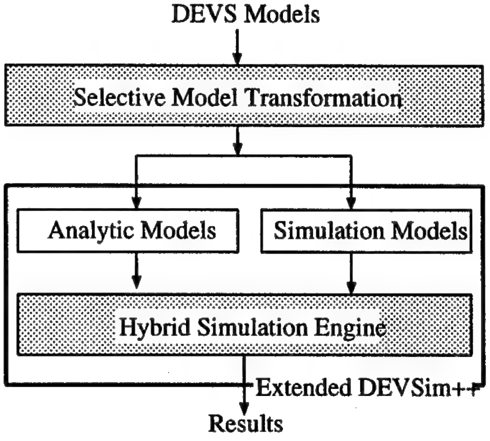


Figure 4: Hybrid Modeling/Simulation with DEVS Models.

3.1 DEVS Formalism

A set-theoretic formalism, the DEVS formalism specifies discrete event models in a hierarchical, modular form. Within the formalism, one must specify 1) the basic models from which larger ones are built, and 2) how these models are connected together in hierarchical fashion.

A basic model, called an atomic model (or atomic DEVS), has specification for dynamics of the model. An atomic model interprets the behavior of a basic component as a state transition machine. An atomic model AM is specified by a 7-tuple[8]:

$$AM =< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$$

X : input events set;
S : sequential states set;
Y : output events set;
δ_{int} : $S \rightarrow S$: internal transition function;
δ_{ext} : $Q \times X \rightarrow S$: external transition function;
$Q = \{(s,e) \mid s \in S, 0 \leq e \leq ta(s)\}$;
$\lambda : S \rightarrow Y$: output function;
$ta : S \rightarrow Real$: time advance function.

The second form of the model, called a coupled model (or coupled DEVS), defines how to couple (connect) several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model, thus giving rise to construction of complex models in hierarchical fashion. A coupled model CM is defined as[8]:

$$CM =< X, Y, M, EIC, EOC, IC, SELECT >$$

X : input events set;
Y : output events set;
M : DEVS components set;
EIC : external input coupling relation;
EOC : external output coupling relation;
IC : internal coupling relation;
$SELECT$: tie-breaking selector.

As proven in [8], the result of coupling DEVS components in a coupled model is itself a atomic DEVS whose state set and input set are cartesian products of all input sets and all total state sets of component models, respectively. Detail descriptions for the definitions of the atomic and coupled DEVS can be found in [8].

3.2 Model Transformation and Simulation Policy

When transforming a simulation model into equivalent analytic one, the transformation should preserve the input/output behavior of the model. For such preservation, we apply the notion of isomorphism or equivalence relation, which is based on one-to-one correspondences between specification structures. If we observe only the input/output behavior of a system, two models are said to be isomorphic or relationally equivalent if input/output behavior of the two cannot be distinguishable in any way[8].

Using the isomorphism, we can represent the steady state behavior of an atomic DEVS model as an equivalent CTMC or as an queueing model. If a model includes a queue for incoming events, it is transformed into a queueing model. Otherwise, an CTMC is used to transform the model.

To set up the isomorphism between a CTMC and the steady state behavior of an atomic DEVS, we make the followings assumptions on atomic DEVS models :

1) the state space of the model is finite;

2) all event types, including internal events, are Poisson process;

3) external and internal transition functions are time-invariant;

4) every state in the state set is reachable from any other states.

Assumption 3 and 4 make the CTMC irreducible and erogodic, thereby solving the CTMC using numerical algorithms. Using the transformed CTMC, steady state probabilities such as mean sojourn time and mean waiting time are obtained. Descriptions on the transformation algorithms can be found in [1].

Analysis of a CTMC with a large state space is very cumbersome and needs vast amount of computation costs. Though a coupled DEVS model can be transformed into an equivalent atomic model, our approach analyzes each atomic DEVS model independently and use the coupling information to merge the transformed CTMCs.

Transformation of DEVS models into queueing models requires some knowledge on the coupling scheme. Actually, two DEVS models, buffer and processor models, are transformed into a queueing network. If we know the rates of incoming jobs of the queueing model, the model acts as a simple delay for the jobs. Otherwise, the rate of the jobs is estimated during the simulation experiments.

Eventually, the transformed analytic models can be represented using the Input/Output Relation Observation(IORO) specification. An IORO observes the behavior of a system in term of input/output relation. It is a structure[8] :

$$IORO =< T, X, \Omega, Y, R >$$

T : time base set;
X : input events set;
Y : output events set;
$\Omega \subset (X, T)$: input segment set;
$R \subset \Omega \times (Y, T)$: I/O relation.

Since the simulation environment manages the simulation time (clock), we just need to describe the three components(X, Y, R) in the structure. As an example of specification, consider a queueing model. It can be represented as :

$$QUEUE_{IORO} =< X, Y, R >$$

$X = \{\text{job}\}$;
$Y = \{\text{out}\}$;
$R : (\text{job,t}) \rightarrow (\text{out, t} + \mu + W)$,

where μ, W are average service time and waiting time in the queue, respectively. In the performance evaluation view point, simulation experiments are the processes of finding the I/O relations of models such as μ, W in the above model.

For hybrid simulation, we combines the abstract simulator algorithms of the DEVS formalism and the analyzers for analytic models. The role of analyzer is to find the average occurrence rates of incoming events and to route the events to the simulators which are responsible for the influencees. To find the influencees, the input and output relations of the associated analytic model are used.

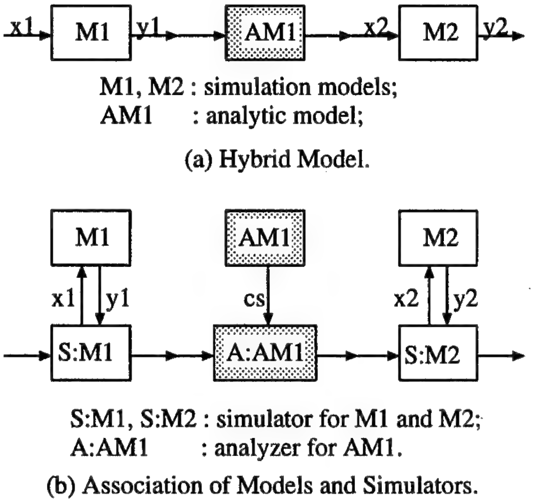

x1
M1
y1
AM1
x2
M2
y2
M1, M2 : simulation models;
AM1 : analytic model;
(a) Hybrid Model.
M1
AM1
M2
x1
y1
cs
x2
y2
S:M1
A:AM1
S:M2
S:M1, S:M2 : simulator for M1 and M2;
A:AM1 : analyzer for AM1.
(b) Association of Models and Simulators.

Figure 5: Hybrid Simulation Strategy.

Figure 5(b) shows the connections between simulators and an analyzer for simulating the model in Figure 5(a). Whenever an event arrives at the analyzer, it counts the arrival to determine the average rate of incoming event. Figure 6 shows the scenario of event flow when an event enters the models M1 of Figure 5.

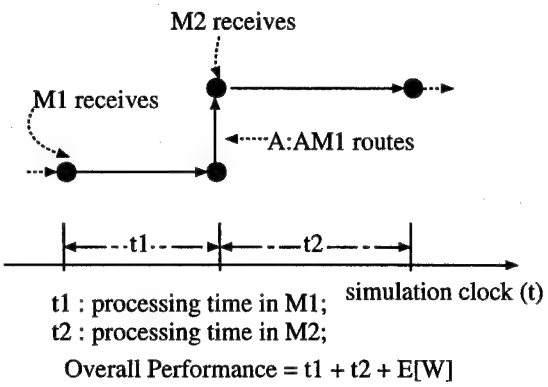

M2 receives
M1 receives
A:AM1 routes
t1
t2
simulation clock (t)
t1 : processing time in M1;
t2 : processing time in M2;
Overall Performance = t1 + t2 + E[W]

Figure 6: Event Flow during Hybrid Simulation.

After a simulation experiment is finished, the analytic model determines the I/O relation in the steady state using the rates. The term E[W] in the performance of Figure 6 is the results from AM1 in Figure 5. When a model knows the occurrence rates of events, the associated analyzer does nothing during the simulation except the routing of events.

3.3 Extended DEVSim++ Environment

To simulate the transformed analytic models with simulation models, we extended the DEVSim++ environment. The original version of DEVSim++[2] realizes the DEVS formalism for modeling and associated abstract simulator concepts for simulation, all in C++. DEVSim++ is a result of the combination of two powerful frameworks for system development : the DEVS formalism and the object-oriented paradigm.

Since DEVSim++ defines classes for modeling and those for simulation separately, it can be easily evolved by developing new classes. Figure 7 shows class hierarchy of the extended environment. The shaded two classes are newly defined for hybrid simulation.

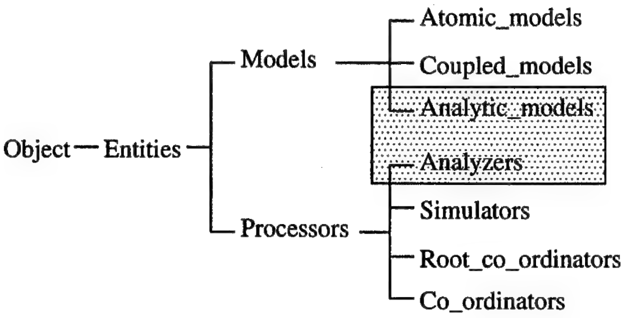


Figure 7: Class Hierarchy of Extended DEVSim++.

The *Analytic_models* class realizes the transformed analytic models. It has instance variables corresponding to three elements in IORO representation : X for input events set, Y for output events set, and R for input/output relations. To manage the I/O relations, we define a structure for input/output relations. *Analytic_models* also defines methods operating on instance variables. *Analyzers* is assinged to *Analytic_models* in a one-to-one manner.

4 Example and Results

As an application of our approach, we consider a simple communication network. As shown in Figure

8, the system consists of a sending node and a receiving node and a set of intermediate routing nodes. The sending node transmits a number of packets to the destination node through the intermediate routing nodes.

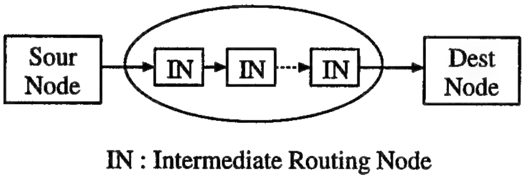


Figure 8: Simple Communication Network.

For comparison of computation time and accuracy of our approach with discrete event simulation, we modeled the system within the DEVSim++ environment. By applying the model transformation method, we can transform the intermediate routing nodes into a single queueing network.

We compare the computation times by measuring the simulation times of both models under the condition that two models generate the same number of packets. All experiments are performed in a Sun Sparc 1+ machine with 32MB main memory.

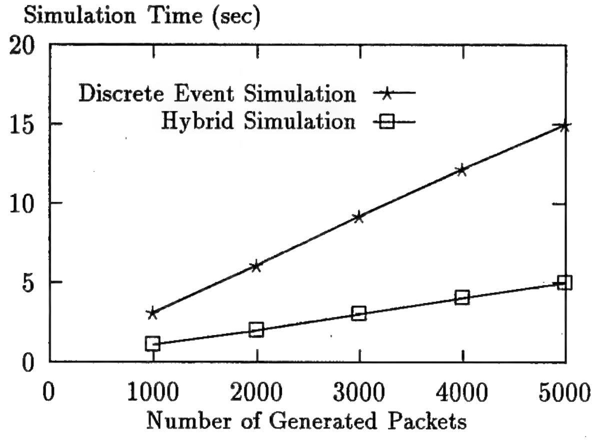


Figure 9: Simulation Time for Varying Number of Generated Packets.

Figure 9 shows the average simulation times when the number of generated packets are varying and systems are configured with two intermediate routing nodes. Each point takes an average of 5 statistically independent simulation runs. The results show that the hybrid approach greatly reduces the computation time.

To compare accuracy between both approaches, we measure the average traveling times of packets, which are the times between the departure at the source node and the arrival to the destination. From the results of Figure 10, we can conclude that there is little difference between two approaches. Though the simulation model is simple, the results shows practical significance of our approach.

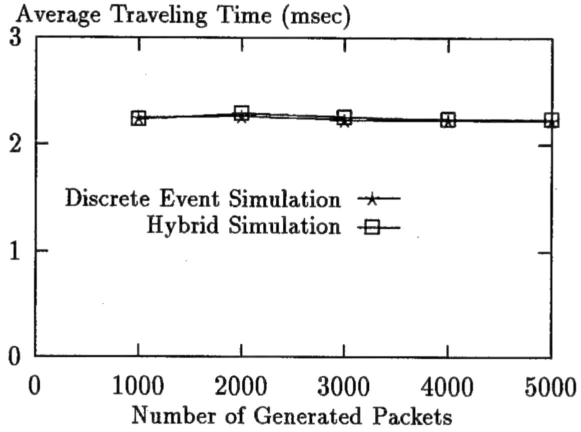


Figure 10: Average Traveling Time of Packets.

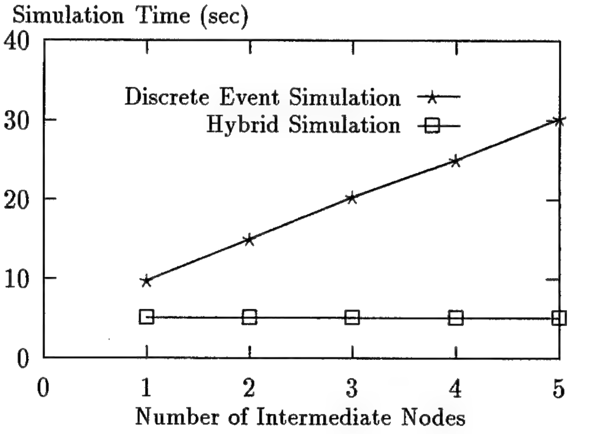


Figure 11: Simulation Time for Varying Number of Intermediate Nodes.

Figure 11 compares the computation costs when the number of intermediate routing nodes are varying. Since the hybrid approach models the tandem-connected nodes as a single queueing model, the computation costs do not increase no longer. But, those for the discrete event simulation continuously increase. From the above results, we can draw a conclusion that our approach has a practical significance, especially when simulating complex systems.

5 Summary and Conclusions

We have proposed a hybrid modeling and simulation framework for high-speed simulation without losing the accuracy of discrete event simulation. The approach is based on a transformation of the steady state behavior of a DEVS model into an equivalent analytic model. For hybrid simulation of the transformed analytic models and DEVS models, we extended the DEVSim++ environment by adding new classes for specifying and simulating analytic models.

Also, we validated the proposed approach by comparing the results with those obtained from simulation experiments with only discrete event simulation models. The results show that our approach can accurately simulate the behavior of a system with greatly reduced simulation time.

Though the approach shows some promising results, there is much work to be done. Currently, an extension of the model transformation method to more general and complex cases is underway. In parallel, we are also extending the DEVSim++ environment to possible automatic model transformation.

Acknowledgements

This work was supported by the Korea Science and Engineering Foundation grant 941-0900-034-2.

References

[1] Myung S. Ahn and Tag G. Kim, "Analysis on Steady State Behavior of DEVS Models", *Proc. of 4th Annual Conf. on AI, Simulation, and Planning in High Autonomy Systems(AIS '93)*, pp. 142 - 147, Sept. 1993.

[2] Myung S. Ahn and Tag G. Kim, *DEVSim++ User's Manual*, Technical Report, TR-CORE-94-1, EE, KAIST, 1994.

[3] Kumar K. Goswami and Ravishankar K. Iyer, "Use of Hybrid and Hierarchical Simulation to Reduce Computation Costs", *Proc. Int. Workshop on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS '93)*, pp. 197-202, Jan. 1993.

[4] L. Kleinrock, *Queueing Systems, Volume II: Computer Applications*, Wiley, New York, 1976.

[5] J.G. Shanthikumar and R.G. Sargent, "A Unifying View of Hybrid Simulation/Analytic Models and Modeling", *Operations Research*, Vol. 31, No. 6, Nov. 1983.

[6] Tag G. Kim and Myung S. Ahn, "Reusable Simulation Models in an Object-Oriented Framework", to appear in *Object-oriented Simulation* (ed: G.W. Zobrist), IEEE Press.

[7] B.P. Zeigler, *Theory of Modelling and Simulation*, John Wiley, NY, 1976(Reissued by Krieger Pub. Co., Malabar, FL. 1985).

[8] B.P. Zeigler, *Multifacetted Modeling and Discrete Event Simulation:* Academic Press, Orlando, FL., 1984.

Session 2C:

DEVS Formalism: Manufacturing Applications

Interface-Oriented Classification of DEVS Models

Carsten Thomas

Research and Technology Department
Daimler-Benz AG
Alt-Moabit 91b, 10559 Berlin, Germany

Abstract

Model classification is a way to structure and handle knowledge about systems. Using the models external message interface as the classification criterion lays emphasis on the fact that atomic and coupled DEVS models are modular and interchangable model components. Interface-Oriented Classification is a means to formalize the conditions of component replacement and coupling valitity. It can be used as the basis of advanced modeling methods like multi-modeling and modeling of structurally variant systems.

1. Introduction

DEVS models contain behavioral and structural knowledge, i.e. knowledge about state variable values, functionality, decomposition into components, and the relations between those components. For the efficient use of models, especially for model re-use and automated model synthesis, also a third sort of knowledge is necessary. This knowledge about common properties of models and the propagation of properties through inheritance is called taxonomic knowledge. By use of taxonomic information, models, which are identical with respect to some property, can be ordered into groups. These groups of models are called classes.

A class of models M_c is a subset of the set of all models M available in some context. The class is denoted by its class name c, which is a member of the set of class names C. Classification of models is done using classification criteria. A classification criterion is a function f_C, which checks the models for the existence of the properties under consideration:

$$f_C\colon \mathrm{M} \rightarrow \mathrm{C} \text{ and } M_c = \{m \in \mathrm{M} | f_C(m) = c, c \in \mathrm{C}\}.$$

There are various classification criteria to be used with simulation models. Most often, an implementation oriented classification is done. In this case, models belong to the same class if their implementation is equivalent. Classification criteria are e.g. whether a DEVS model is an atomic or coupled model or what state variables are used.

For most of the application areas for classification the implementation has no significance. Instead, we need to know about which incoming external events a model can process, which outgoing external events a model produces and how it can be coupled to other models. These properties are what we call the models interface. In the following sections, we propose a classification scheme where the models interface is used as a classification criterion. We call this scheme the Interface-Oriented Classification of DEVS models.

2. A brief look at the DEVS formalism

Applying the DEVS formalism, models are constructed in a modular hierarchical manner [6,8]. Atomic models describe the functionality of basic system entities. Coupled models correspond to more complex system entities. The inner structure of the complex entities are represented by the components of the coupled model and their coupling. Both, atomic and coupled models can be used as model components.

Atomic DEVS models describe the dynamics of a system by means of input and output sets X and Y, a set of sequential states S, state transition functions δ, an output function λ and a time advance function τ:

$$M = \langle X, Y, S, \delta, \lambda, \tau \rangle.$$

Coupled DEVS models consist of a set of component names D and the corresponding set of models $\{M_i\}$. The coupling of every single component is specified by a set of influencees I_i, and a set of output/input relations $Z_{i,j}$ for every component/influencee pair. A tie-breaking select function ς resolves event processing conflicts among the components:

$$N = \langle D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, \varsigma \rangle.$$

Usually, ports are used to structure the input and output sets of models [2]. Then, the input and output sets of a model are structured sets denoted by

0-8186-6440-1/94 $04.00 © 1994 IEEE

$$X \subseteq \{(n,v) | n \in N^X, v \in V\} \text{ and}$$
$$Y \subseteq \{(n,v) | n \in N^Y, v \in V\},$$

where N^X and N^Y are the sets of input and output port names and $v \in V$ is additional information contained in the messages which represent the external events.

When ports are used, the output/input relations of a model N can be specified in the form

$$Z_{i,j} \subseteq \{((i,n_i),(j,n_j)) | i \in D \cup d_N, j \in I_i, n_i \in N_i^Y, n_j \in N_j^X\},$$

where d_N is the identifier of the coupled model [1].

In this approach, no statements can be made about the validity of a coupling specification, i.e. there is no chance to ensure that a model is capable of processing the messages it receives from its predecessors. Also, nothing can be said about whether a model can replace another model in a given coupling specification. With Interface-Oriented Classification of DEVS models we address these problems.

3. Model interfaces

Expanding the idea of structured input and output sets, we suggest to use typed messages. Then, a message is denoted by a tri-tuple (n,u,v), consisting of a port name n, a message type name u, and an additional information item v, whichs value is within the range of the type u. Using this notation, we are able to describe the input and output sets of a model as

$$X = \{(n,u,v) | n \in N^X, u \in U_n, v \in V_u\} \text{ and}$$
$$Y = \{(n,u,v) | n \in N^Y, u \in U_n, v \in V_u\}$$

where N^X is the set of names of the input ports of that model, and N^Y is the output port name set. We apply the restrictions that

(i) types with identical names have identical ranges and
(ii) the range of a given type is constant over time.

Due to these restrictions, the incoming messages a model can process, and the outgoing messages a model produces are fully described by the structure

$$I = \langle P^X, P^Y \rangle,$$

where $P^X = \langle N^X, \{U_n | n \in N^X\} \rangle$ is the input port set and $P^Y = \langle N^Y, \{U_n | n \in N^Y\} \rangle$ is the set of output ports. The structure I is called the models interface.

3.1. Interfaces of atomic models

Since there is currently no standardised way of formal definition for the functionality of a model, namely for the external event and output functions, we are not able to extract the interface description automatically for atomic models. Instead, this description has to be given by the implementer of the model.

Consider the model of a machine tool which processes workpieces (figure 1a). Workpieces are a message type WP, and its range is $\{raw, processed\}$. Then, the interface of the machine tool is $I_{MT} = \langle P_{MT}^X, P_{MT}^Y \rangle$ with $P_{MT}^X = \langle \{in\}, \{\{WP\}\} \rangle$ and $P_{MT}^Y = \langle \{out\}, \{\{WP\}\} \rangle$.

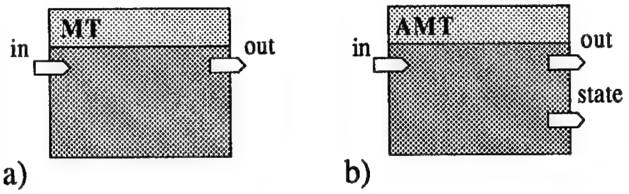


Figure 1. Atomic models of a machine tool MT (a) and an advanced machine tool AMT (b)

A more advanced machine tool could have an additional binary output port indicating the state of its workpiece processing buffer after every event (figure 1b). The interface of this advanced machine tool would be $I_{AMT} = \langle P_{AMT}^X, P_{AMT}^Y \rangle$ with $P_{AMT}^X = P_{MT}^X$ and $P_{AMT}^Y = \langle \{out, state\}, \{\{WP\}, \{Bool\}\} \rangle$.

3.2. Interfaces of coupled models

The interface of a coupled model is defined by the interfaces of its components. Therefore, the port characteristics (and thus, the interface) of a coupled model can be computed automatically if the interfaces of all components and their coupling are given.

The set of message types, which can be accepted by an input port of a coupled model, is the intersection of the message type sets of all component input ports connected to that port:

$$U_n = \bigcap U_{n_i} \ \forall n_i \Big| ((d_N, n_{d_N}),(i,n_i)) \in Z_{i,j}, n_{d_N} \in N_N^X, n_i \in N_i^X.$$

The message type sets of the input ports are used to specify the input set of the coupled model:

$$X = \{(n,u,v) | n \in N^X, u \in U_n, v \in V_u\}.$$

If the message type sets of the connected ports are disjunct, the message type set of the coupled models input port is empty. This indicates a modeling error.

The set of message types which can be sent out by an output port of a coupled model can be computed in the same way from the message type sets of the connected component output ports. In this case, the result set is the union of the component port message type sets:

$$U_n = \bigcup U_{n_i} \ \forall n_i \Big| ((i,n_i),(d_N, n_{d_N})) \in Z_{i,j}, n_i \in N_i^Y, n_{d_N} \in N_N^Y.$$

The output set of the coupled model then is

$$Y = \{(n,u,v) | n \in N^Y, u \in U_n, v \in V_u\}.$$

Consider the model of a buffered machine tool, consisting of an advanced machine tool as shown in figure 1b and a buffer in which unprocessed workpieces are stored (figure 2).

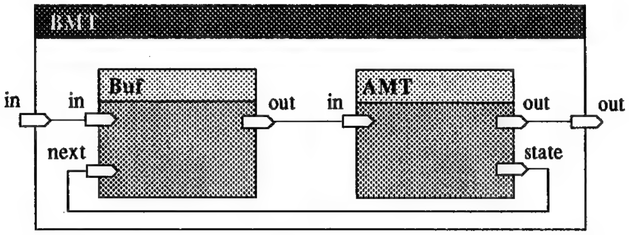


Figure 2. Coupled model of a buffered machine tool BMT

Let the interface of *Buffer* be $I = \left\langle P^X_{Buf}, P^Y_{Buf} \right\rangle$ with $P^X_{Buf} = \left\langle \{in, next\}, \{\{WP\}, \{Bool\}\} \right\rangle$ and $P^Y_{Buf} = \left\langle \{out\}, \{\{WP\}\} \right\rangle$. The interface of the buffered machine tool model can be computed from the interfaces of *Buffer* and *AMT*. The result is $I = \left\langle P^X_{BMT}, P^Y_{BMT} \right\rangle$ with $P^X_{BMT} = \left\langle \{in\}, \{\{WP\}\} \right\rangle$ and $P^Y_{BMT} = \left\langle \{out\}, \{\{WP\}\} \right\rangle$.

3.3. Relations over model interfaces

We can formally define relations over model interfaces as a step to define the classification criterion for Interface-Oriented Classification.

The equivalence of two models is defined by the equivalence relation $I_A = I_B$.

$$I_A = I_B \quad \text{iff}$$
$$N^X_A = N^X_B \quad \wedge \quad U^A_n = U^B_n \ \forall n \in N^X_A$$
$$\wedge$$
$$N^Y_A = N^Y_B \quad \wedge \quad U^A_n = U^B_n \ \forall n \in N^Y_A$$

This says, that the interfaces of two models are equivalent, if the available in- and output ports have identical names, and for every single of these ports identical message type sets are defined. It is assumed, that message types with identical names have identical ranges.

The ordering relation for the interfaces of two models is defined as

$$I_A \le I_B \quad \text{iff}$$
$$N^X_A \subseteq N^X_B \quad \wedge \quad U^A_n \subseteq U^B_n \ \forall n \in N^X_A$$
$$\wedge$$
$$N^Y_A \subseteq N^Y_B \quad \wedge \quad U^A_n \supseteq U^B_n \ \forall n \in N^Y_A, n \in P^Y_B:$$

This says, that the interface of model A is contained within the interface of model B, if the B-interface at least contains the names of all ports of A. For all the input ports of B, at least the message types defined for the corresponding ports of A must be supported; none of the output ports of B corresponding to an output port of A is allowed to use message types not defined for the A-port.

4. Classification and inheritance

By using the relations defined for model interfaces, we establish the classification criterion for the Interface-oriented Classification. Two models belong to the same class M_c, if their interfaces are equivalent:

$$A, B \in M_c \Leftrightarrow I_A = I_B = I_c.$$

Since the interfaces of models belonging to a class are equivalent, they also denote the interface I_c of the class itself. In the example, the interfaces of the simple machine tool and the buffered machine tool are equivalent. They form a class of models $classMT = \{MT, BMT\}$, while the advanced machine tool is member of another class $classAMT = \{AMT\}$.

Between the classes, interface oriented inheritance relations can be defined. A class inherits the interface of another class, if its interface contains the interface of the predecessor class as described by the ordering relation $I_{classA} \le I_{classB}$. So, the class of advanced machine tools *classAMT* inherits the interface of the simpler machine tools *classMT*, which is then extended by the state port. It can be shown, that interface inheritance also works for multiple base classes.

Class trees are a graphical representation scheme for inheritance information. In the class tree structure, the nodes are labelled with class identifiers, and the directed arcs represent "initial_node is_base_class_of terminal_node" relations. In figure 3, *classMT* is shown to be the base class of *classAMT*, since *classAMT* inherits the interface of *classMT*.

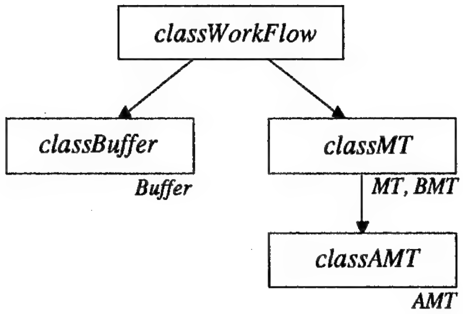


Figure 3. Example of a model class tree

In addition to the class hierarchy imposed by the equivalence and ordering relations, the inheritance tree can be further structured to emphasise functional differences between classes. In figure 3, the interfaces of *classWorkFlow* and *classMT* are equivalent. However, *classMT* is a special work flow element which shouldn't be confused with others, so machine tools form a separate class.

5. Application areas

5.1. Model construction and management

With Interface-Oriented Classification, support for model construction and model management can be significantly enhanced.

Ensuring syntactical correctness: Models are considered to be syntactically correct, when the components are able to process all incoming messages. When using typed messages, it can be ensured that only ports with matching message type sets are connected to each other. Syntactical correctness of the model is guaranteed when strict matching is ensured, i.e. when

$$U_{n_i} \subseteq U_{n_j} \ \forall n_i, n_j \Big| \big((i,n_i),(j,n_j)\big) \in Z_{i,j} .$$

In figure 2, this condition is fulfilled e.g. for the connection between output port *Buf.out* and input port *AMT.in*, since the port message type sets are equal: $\{WP\} = \{WP\}$.

However, most often it is not practicable to enforce this restriction. A weaker version of the condition can be used to test whether there is at least a possibility that two communicating components will understand each other:

$$U_{n_i} \cap U_{n_j} \neq \varnothing \ \forall n_i, n_j \Big| \big((i,n_i),(j,n_j)\big) \in Z_{i,j} .$$

Testing for interchangeability: Models can replace other components in coupled models without a change of the coupling, if the replacing model has an equal or more extensive functionality regarding the interface. This requirement is fulfilled if the replacing model

(i) has at least the ports of the replaced component, so that it can be coupled to the surrounding structure in the same way,

(ii) is able to process all the message types on the used input ports, which the replaced component was able to process, and

(iii) sends only message types on the used output ports, which also could have been sent by the replaced component.

All these conditions are satisfied, if the replacing model belongs to the same class as the replaced component or a class derived from that class.

A model of class *classAMT* can replace a component of class *classMT*, since it fits well into the given coupling (figure 4). The input port *AMT.in* accepts the same messages as *MT.in*, and *AMT.out* doesn't send anything what might confuse components coupled to that port. The messages sent from port *AMT.state* are not used because the port isn't used.

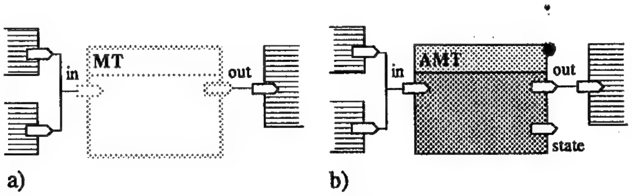


Figure 4. Replacing a model component (a) with a model belonging to a derived class (b)

Supporting SES construction: Interface-Oriented Classification can also be used in combination with the *System Entity Structure* (SES), a knowledge representation scheme for model families introduced by Zeigler [6,8]. There are two obvious applications in this field.

An aspect node is associated to a coupling specification for the decomposition it represents. The syntactical correctness of this specification can be proved in the way described above.

A specialization node is followed by entities which are specialized variants of the predecessor entity of this node. When the variants are members of the same class as the predecessor entity or a class derived from that class it is ensured that all variants fit into couplings where the preceding entity would fit in. This can be used to partially prove the consistency of a SES.

5.2. Advanced modeling methods

Interface-Oriented Classification is the basis of our approach to advanced modeling methods like multi-modeling and modeling of structurally variant systems.

Multi-modeling: Multi-modeling is a technique which allows to combine different means of description within one model. In multi-models, a subsystem can be modeled using the formalism which fits its nature best [3]. Especially graphically representable formalisms like queuing systems and Petri nets are combined in known approaches.

In our approach, the basic elements of other means of description are mapped onto DEVS components. For instance, sources, servers, and sinks in queuing models are represented by atomic or coupled models. The integrity of coupled models using such "emulated" components can be ensured by methods of the Interface-Oriented Classification.

Consider a special coupled model which is used to emulate a queuing model. Then, to ensure model integrity, only queuing model components like sources, queues, servers, and sinks are allowed. This restriction can be met automatically by checking whether the components belong to the respective classes or to successor classes.

Since Interface-Oriented Classification does not depend on implementation issues, atomic *and* coupled models can be used to "emulate" the basic components of other formalisms. The coupled models emulating such components in turn can be refined using just another means of description. Therefore, model refinement using other formalisms is not limited to one level of aggregation.

Modeling of structurally variant systems: Often, systems to be modeled undergo structural changes: Components are assembled to form products, workers supervising a machine tool come and go, control strategies of machines are exchanged. Known techniques to model such changes often map structural changes to state changes or directly interfere with the state or structure of the model to be changed, violating its modularity [5,7].

Prerequisite for our approach to model structurally variant systems is a method to instantiate fully defined atomic or coupled models at simulation time and to delete them after use. The models instantiated by such a method are not statically connected to the model structure and thus are named "free" models. Information about such free models can be transmitted between components by messages, and can be stored in state variables of atomic models.

For the modeling of structure-changing systems, a facility is needed to dynamically link these free models to the static model structure. Applying the methods of Interface-Oriented Classification, a special form of coupled model, called model variable, can be used for this.

Model variables are placeholders in static model structures. They are either empty or filled with one atomic or coupled component (figure 5). When a component residing a model variable is replaced by another variant of this component, the entire model changes its structure.

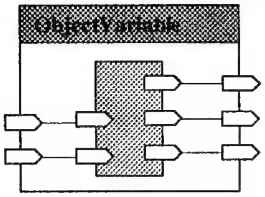


Figure 5. Object variable dynamically linked to a component

Model variables must have a invariant interface, so that they can be linked to the static model structure. To define the interface of a model variable, it has to be assigned to a class. Then, the model variables interface is equivalent to the interface of the class: it gets all the ports the member models of the class have, and the ports accept or send the message types as defined for the class.

When an information about a free model is received by the co-ordinator of the model variable, it checks whether the class of the free model is identical to or derived from the class of the model variable. If this restriction is met, the co-ordinator establishes links between the ports of the model variable and the identically named ports of the free model, thus dynamically linking it into the static model structure. The dynamic links are cut when either a new free model has to be linked in or special control messages are received by the co-ordinator.

Using interface and class information from the Interface-Oriented Classification, structural changes of systems can be easily modeled without violating the modularity principle of the formalism.

6. Conclusion

Interface-Oriented Classification is a classification scheme which emphasizes the modularity and encapsulation principles of the DEVS formalism. With the information available through this technique, the modeling process can be supported during coupled model specification and SES building process. Furthermore, Interface-Oriented Classification is the formal basis for our approaches to advanced modeling techniques like multi-modeling and modeling of structure-changing systems.

Interface-Oriented Classification methods have been implemented and used in a prototypical modeling and simulation system currently under development at Daimler-Benz [4]. While current applications concentrate on the design and modeling of systems, further research will focus on the utilisation of Interface-Oriented Classification in the operation of model-based autonomous intelligent systems.

References

[1] Chow, T. H.: *A Hierarchical, Modular Simulation Environment for Flexible Manufacturing System Modeling.* Dissertation, University of Arizona, Tucson (AZ), 1993

[2] Linvy, M.: DELab - A Simulation Laboratory. *Proceedings of the 1987 Winter Simulation Conference.* SCS Publications, San Diego (CA), 1987, 486 - 494

[3] Miller, V. T.; Fishwick, P. A.: Graphical Modeling Using Heterogeneous Hierarchical Models. *Proceedings of the 1993 Winter Simulation Conference.* SCS Publications, Los Angeles (CA), 1993, 612 - 617

[4] Thomas, C.: Hierarchical Object Nets - A Methodology for Graphical Modeling of Discrete Event Systems. *Proceedings of the 1993 Winter Simulation Conference.* SCS Publications, Los Angeles (CA), 1993, 650 - 656

[5] Uhrmacher, A. M.: Variable Structure Models: Autonomy and Control – Answers from Two Different Modelling Approaches. *Proceedings of the 4. Annual Conference on AI, Simulation and Planning in High Autonomy Systems.* IEEE CS Press, Tucson (AZ), 1993, 133-139

[6] Zeigler, B. P.: *Multifaceted Modelling and Discrete Event Simulation.* Academic Press, London, 1984

[7] Zeigler, B. P.; Prähofer, H.: Systems Theory Challenges of Variable Structure and Intelligent Systems. in: Pichler, F.; Moreno-Diaz, R.: *Computer Aided Systems Theory – EUROCAST '89.* Springer, New York, 1989

[8] Zeigler, B. P.: *Object-oriented Simulation with Hierarchical, Modular Models – Intelligent Agents and Endomorphic Systems.* Academic Press, Boston (MA), 1990

Generation, Control, and Simulation of Task Level Actions Based on Discrete Event Models

J.M. Couretas J.W. Rozenblit

Department of Electrical and Computer Engineering
The University of Arizona
Tucson, Arizona 85721
{couretas,jr}@ece.arizona.edu

Abstract

A model based method for task level command generation is used here to simulate a pipeline process. Using a discrete event systems formalism, a method for describing manufacturing systems is reviewed prior to constructing the sequential assembly line simulation. This precedes a study of the entire assembly line model, its coordination, and performance metrics. An example system is then simulated and analyzed for language generation and system performance.

1 Introduction

Expanding computer power is causing increased interest in modeling and simulation. This is especially true in engineering where physical prototypes can be costly and time consuming. One step in this direction was taken by Zeigler [3] in implementing the Discrete Event System Specification (DEVS) in a software environment called DEVS-Scheme [2]. It facilitates the construction of modular, hierarchical models and their organization.

Jacak and Rozenblit [1] introduce the exact form robot and workstation models take to work together in an assembly line simulation. Their method deals with a series of workcells, each of which has its own processing program. Moving parts between the workstations is done by robot pick and place actions. A program of robot and workstation instructions is then expressed in a task-oriented robot programming language [4] [5] (TORPL). This program controls a discrete event system simulating the sequential manufacturing process that is constructed using this methodology. It is then monitored for performance.

An additional layer is introduced for motion planning to their manufacturing system representation. This provides collision free geometrical path planning and optimal trajectory planning.

2 Background

2.1 Discrete Event Systems Representation

With the overall goal of achieving a means for rapid modeling and simulation of the entire technological line, a representational formalism is required. In this case, we employ the Discrete Event System Specification (DEVS) formalism [3].

DEVS specification consists of external inputs, X, external outputs, Y, and a set of states, S. States transitions are due to either an external event or the time limit, t_a(s), elapsing. A state change due to an elapsed time limit is called an internal transition, δ_{int}. Similarly, state changes due to external events are called external transitions, δ_{ext}. All of these transitions occur over the system's state set, S.

Interpretation of the DEVS and a full explication of the semantics of the DEVS are in [3].

2.2 Manufacturing Environment Representation

The system presently under consideration is a sequential technological process. This consists of robots and workstations where robots do pick and place actions between workstations. Robot actions in this system occur so that each step has an associated set of instructions in the task oriented robot programming language [1], hereafter referred to as TORPL.

The basic macroinstructions of TORPL are:

$$MOVE \left\{ \begin{array}{l} EMPTY \\ HOLDING \end{array} \right. \quad TO \text{``}position\text{''}$$

0-8186-6440-1/94 $04.00 © 1994 IEEE

PICKUP “part” AT “position”
PLACE “part” AT “position”
WAIT FOR “sensor input signal”
START “output signal”

The above instructions synthesize the robot’s action program. This process requires an introduction of conditional instructions that depend on the states of each device of the assembly line. Thus, defining a program simulator requires modeling conditions that enable each program instruction. An example assembly line is shown in the Figure 1.

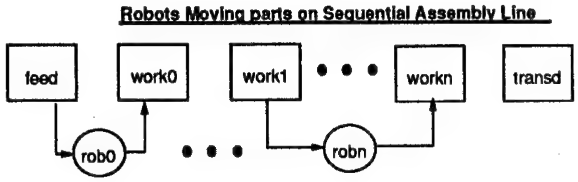


Figure 1: Assembly Line

2.3 Experimental Frame

The experimental frame executes and monitors a model. It consists of a generator sending inputs to a model and a transducer that observes the resultant model behavior. This concept is implemented on the example assembly line by making the generator a parts feeder and the last workstation on the assembly line a transducer. Feed rate of incoming parts is modulated with the internal transition time of the feeder, and assembly line performance is observed with the transducer.

The feeder, as shown in Figure 2, feeds parts to the assembly line at given time intervals. Upon sending a part to the assembly line, the feeder waits in a passive state for the specified time period before sending another part.

The transducer is positioned as the last workstation in the assembly line. Here, it takes in parts as they are completed and performs user specified calculations. The transducer is also responsible for maintaining the observation time of the overall simulation. Once this observation time is exceeded, all models stop work. The transducer is shown in Figure 3.

2.3.1 Robot Definition

As explained in [1], robot states reflect position and action. Robot position designates where it is on the simulated assembly line. Robot actions include picking up a part, placing a part, or doing nothing.

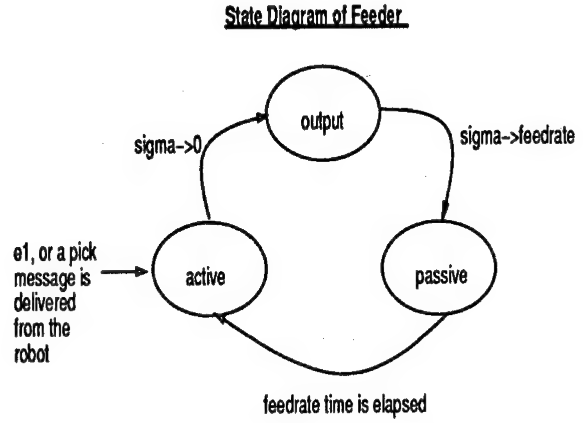


Figure 2: State Behavior of Feeder

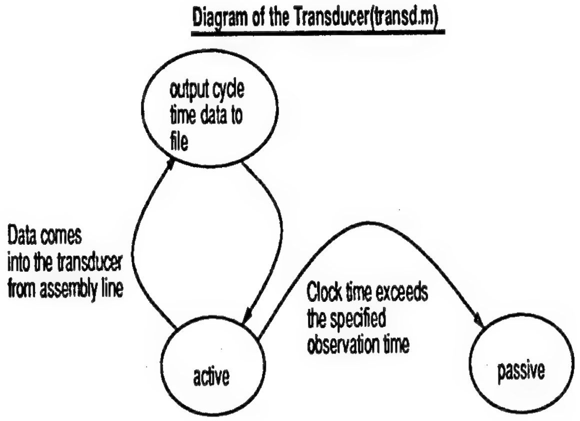


Diagram of the Transducer(transd.m)
output cycle time data to file
Data comes into the transducer from assembly line
Clock time exceeds the specified observation time
active
passive

Figure 3: State Behavior of transducer

2.3.2 Robot Operation

Zeigler [2] introduces event-based control as a method of monitoring an operation. This architecture consists of two models, a sender and receiver. The sender has a time window in which it expects the receiver to confirm the sent command’s completion. Should the receiving model’s confirmation not return within this time window, the sender assumes failure.

Event based control is used here to interpret robot tasks as a *state-space* [6] representation of actions needed to complete the task. The event based controller, hereafter referred to as EBC, controls the robot by sending subsequent actions, upon confirmation that the previous action is done, until the assigned task is complete. This is shown in Figure 4.

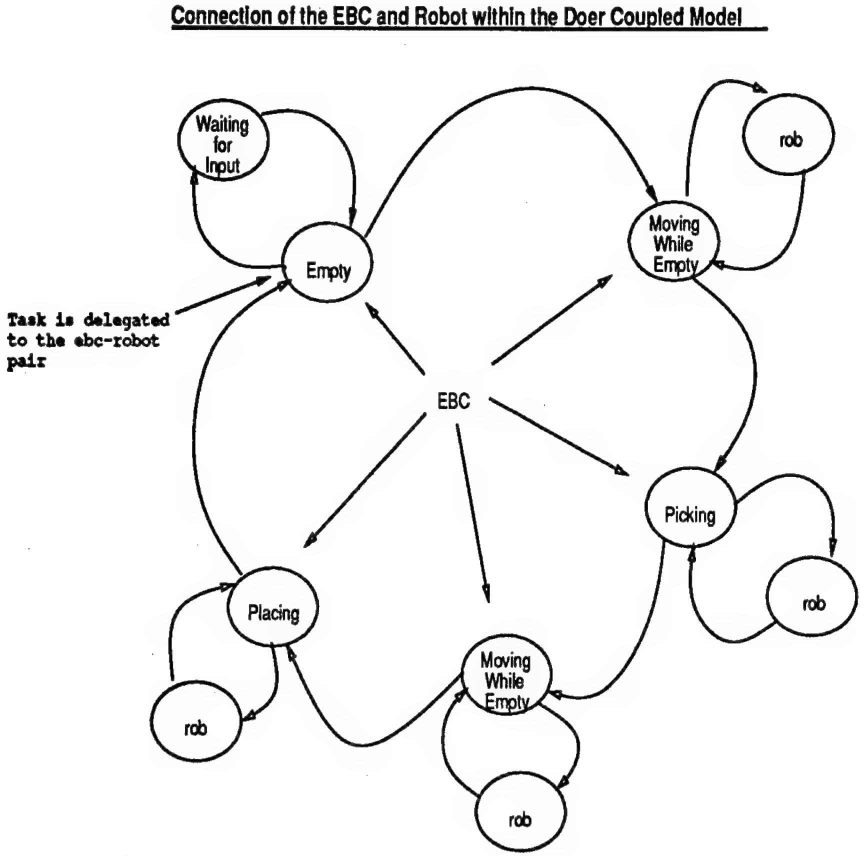

Connection of the EBC and Robot within the Doer Coupled Model
Waiting for Input
Empty
Task is delegated to the ebc-robot pair
EBC
rob
Moving While Empty
Picking
rob
Placing
rob
Moving While Empty
rob

Figure 4: Internal Robot Behavior

2.3.3 Workstation Definition

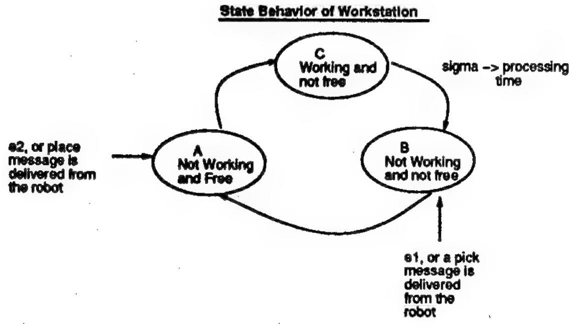

State Behavior of Workstation
C Working and not free
sigma -> processing time
e2, or place message is delivered from the robot
A Not Working and Free
B Not Working and not free
e1, or a pick message is delivered from the robot

Figure 5: Internal Workstation Behavior

Workstations have three states. These states are "not working and free," "not working and not free," and "working and not free." This is defined in [1].

The workstation transitions between these states by interacting with the robot. For example, a workstation in state "not working and free" which gets a part "placed" on it by a robot goes to state "working and not free." The workstation will automatically transition to state "not working and not free" upon completing the part. Robot "picking" is then required to return the workstation to the state "not working and free". This is shown is Figure 5.

3 Modeling Effort

3.1 System Construction

Having decided on the methodology and model forms, the next step is building the sequential assembly line simulation. This includes constructing and testing each model independently, organizing models into an overall structure, coupling them, and deciding on performance metrics.

Model construction was software implementation of the sequential assembly line. With the models developed, the next step was aggregating them into one overall structure and coupling them. Communication between the models was achieved through name directed coupling. This consists of sending communications directly to the model via its name.

3.2 System Architecture

The system architecture is centralized. This means that one central module, the controller, processes all of the assembly line entity states and positions, and then decides on work allocation. The system simulator's architecture is shown in Figure 6.

3.3 Task Formation

The controller's goal is to complete all tasks presently in the system. Tasks needing attention are determined by the states of sequential workstations. Robot and workstation states are communicated to the controller upon every change.

The controller is a central data repository into which state updates for all assembly line entities are directed and stored. Workstation states are tracked by a service list. It is called a service list because the workstations are continually monitored for potential requirement of service. Other lists used are the task list, robot list, and job list.

The service list contains the states of all workstations. The service list maintains a state for every workstation on the assembly line, all the time. When state updates come in from the assembly line, the service list is updated to the new assembly line representation. An example of the service list is shown below:

service-list = ((0 b)(1 a)(2 c)(3 a) ...)

A task is formed whenever two workstations are sequentially in states "B" and "A". When this happens, the two addresses are combined to make a task as shown below:

States:	(1 b)(2 a)
Task:	(1 2)

A task list could have any number of tasks, and would take the following form:

Task-list: ((1 2)(3 4) ...)

An example robot list has the following form:

robot-list = ((rob1 move-while-holding) ...)

Multiple tasks require multiple robots to maximize system performance. This need is met by allowing any number of robots to service the assembly line.

Once tasks are formed, a search is made for the closest robot to perform the task. Proximity is measured both by physical distance and by an estimation of time to completion from the present task. Physical distance is the distance from the present location of the robot to the first workstation in the given task. This is what is used for free robots - robots that are empty and waiting for an assignment. If a robot is still completing an action, distance equivalent approximations are interpolated from the present robot state. This approximation is then added to the actual distance between addresses in order to find the total distance between the present robot and the first workstation in the task.

Robots presently working on a task are also evaluated for their proximity to tasks awaiting assignment. The main reason for this is that a robot close to a task to be assigned could be nearly finished with its present work when the nearby task becomes available. If only idle robots were considered, an idle robot farther away might be chosen for the task, resulting in a high travel time. Choosing the robot already working amounts to not choosing a robot, and waiting until that robot reports that it is free before assigning the task.

When a task is assigned to a robot, this becomes a "job." Jobs are sent to the robot designated as the first element of the job. An example of a job is shown below:

Available Robot:	(rob1)
Task:	(1 2)
Job:	(rob1 (1 2))

3.4 Robot Application Strategy

An example of control modification is assigning different location prioritization algorithms to the robots. They are assigned to prioritize service to the beginning of the assembly line, the end, and the closest available task. This exercise monitors the system performance differences of alternative prioritization schemes.

Priority on the beginning and end of the assembly line started off as an algorithmic exercise. Over time, however, we saw the merits of either of these in application. Prioritizing the beginning of the assembly line might occur in situations where the line is, for one reason or another, perpetually starved for parts. Similarly, priority on the end of the assembly line, on getting parts out, might occur in a situation where there is a bottleneck at the end of the assembly line. Additionally, on a multi-robot assembly line, different robots could be assigned different operational

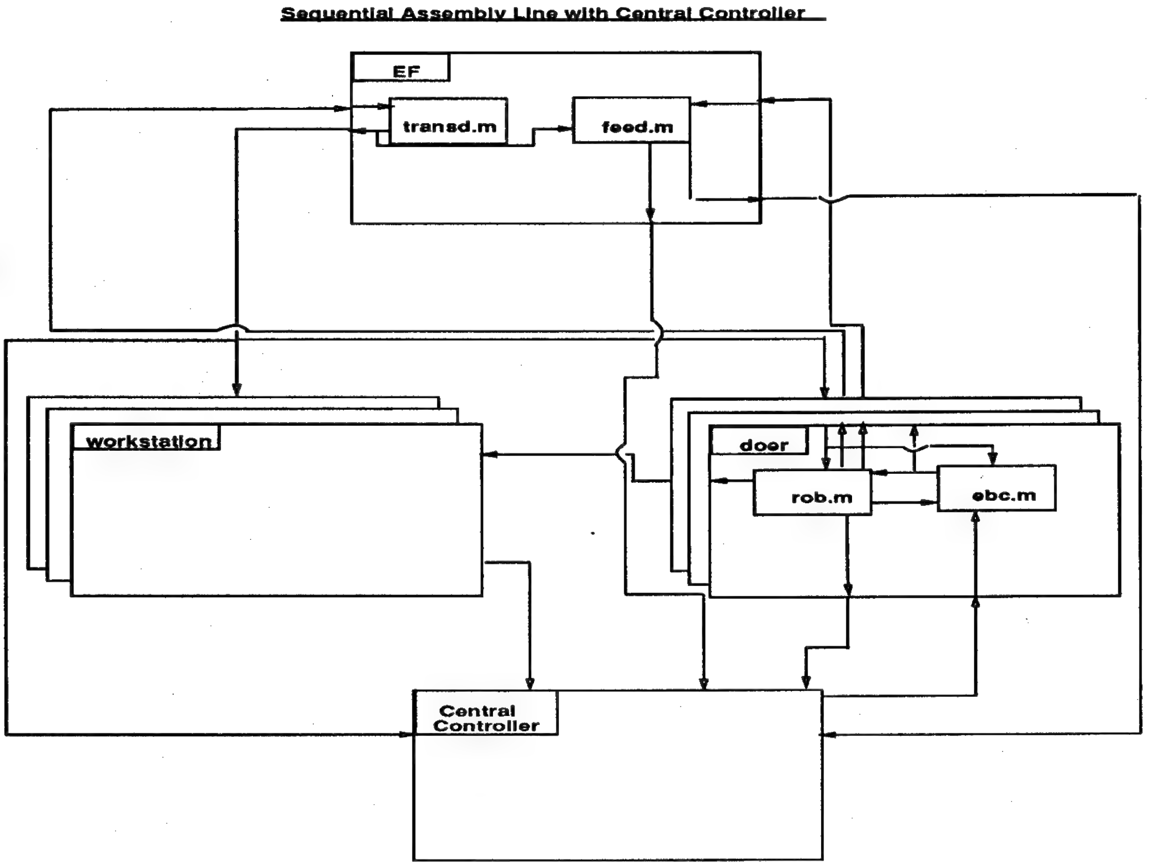


Figure 6: Assembly Line Architecture

algorithms for part processing.

Simulating the robots with different prioritization schemes gives us a trace of the robot movements on the sequential assembly line. Assuming the simulation accurately depicts the assembly line, this "trace" is validated robot code that could be used to control the actual robots working on an assembly line.

3.5 TORPL Code

TORPL code is sequentially output for each step taken by the robot model. A task, (0 1), assigned to $robot_1$ generates the following TORPL code:

$robot_1$ **move while empty**	$robot_1$ moves to position 0.
$robot_1$ **pickup widget at feed**	$robot_1$ picks up the part at position 0, the feeder.
$robot_1$ **move while holding**	$robot_1$ moves the part between workstations 0 and 1.
$robot_1$ **place widget at 1**	$robot_1$ places the widget at workstation 1.

This process is carried out in detail for each task performed. Completing the tasks leads up to completing entire jobs, and job completion leads us to look at system performance.

4 Assembly Line Performance Evaluation

In this simulation model, assembly line throughput was selected as the performance measure. It was calculated on-line by a transducer that totals the number of parts completed and divides this by the total time that the line is operational.

This measurement is similar to what one finds on actual assembly lines. The system does an on-line update of throughput, and this allows any internal control methods that rely on throughput to automatically evaluate the present state of the system and take action.

4.1 Simulation Setup

Setting up the simulation requires deciding the feed rate of parts coming into the process, the mean processing time per workstation, and the robot execution times. The feedrate, controlled by the feeder, is one new part into the process every 5 time units. Workstation processing time is a uniform distribution from 1 to 10 time units. The robot takes 2 time units to pick up or place a part, while moving takes the same number of time units as the distance traversed. In this system, 1 distance unit equals 1 time unit.

4.2 Analysis of Model Results

Looking at throughput data for the 5000 time unit production run in Figure 7, we see the following throughput rates:

Robot focusing on the beginning	0.0492
Robot focusing on closest task	0.0494
Robot focusing on the end	0.0455

While the above numbers are relatively close, the robots prioritizing the closest available task and on the beginning of the assembly line exceed the performance of the robot prioritized to the end of the assembly line.

This performance discrepancy by the robot focusing on the end of the assembly line is due to its increased travel time between priority workstations at the end of the line and jobs earlier in the line. The robot focusing on the closest task has minimal travel time. The robot focusing on the beginning of the assembly line benefits from each workstation in the assembly line holding a completed job before it moves to the end of the line.

4.3 Data Analysis

At 0.0494 jobs per time unit, the robot prioritized to the closest task was slightly better than the robot prioritized to the beginning of the assembly line at 0.0492 jobs per time unit. A less intuitive merit of the robot prioritized to the beginning is that every workstation sits with a completed job before it starts completing jobs. So many jobs at the end of the line results in"spurts" of job completions, exemplified by throughput oscillations as shown in Figure 7.

The slowest throughput, 0.0455 jobs per time unit, came from the robot prioritized to the assembly line's end. Decreased performance with this algorithm results from its priority on getting jobs to the assembly line's end. It uses too much travel time going from end of the line jobs to those earlier in the process.

Algorithms focusing on the beginning of the assembly line, or simply on the closest task at hand, outperform the algorithm whose only goal is output. A focus on the process, instead of only the goal, turned out to be the winning method.

5 Future Work and Conclusions

Model operation exemplified how TORPL code can be generated and verified before implementation in an actual manufacturing system. The example here requires both task formation, and assignment of the task to the robot most suited. Real world scenarios are rarely so simple. Future systems require planning to deal with unprecedented difficulties.

Planning is commonly approached in two ways, on-line and off-line. Off-line planning has merit in that computation time does not impose upon the present process. On-line planning benefits the user in its autonomy. Decisions are made and executed on the spot.

ElMaraghy and Rondeau [7] propose "a new environment for off-line programming of robot tasks, including a feature-based geometric database, an off-line programming system with a knowledge base, an expert task and motion planner, and a run-time monitoring system." This comprehensive planning methodology would substantially extend control capability in our system.

Extending task formation in our system to include the benefits of planning is a natural extension. Similarly, extending individual or group robot capability would take us from pick-and-place robotization to a myriad of possibilities.

Decentralization of this system would involve moving task and job formation to the robot level. This is similar to Jacak and Rozenblit's [1] original conception of attaching an acceptor to each robot in order to allow it an immediate state representation of the assembly line. The difference will be transforming robots into endomorphic agents as conceived by Zeigler [2].

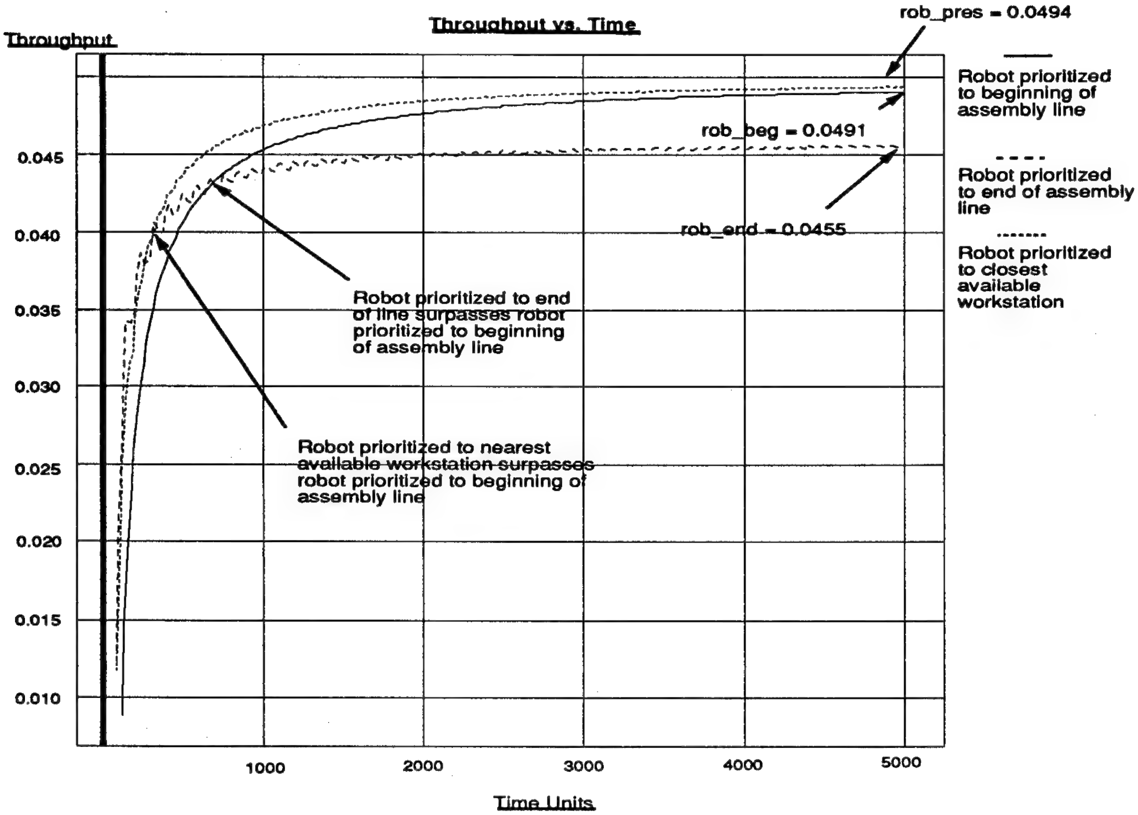


Figure 7: Assembly Line Throughput Performance

Adding planning and decentralization to assembly line entities opens up a whole new world of robustness for the system. Changing robots from simple slaves of the centralized controller to autonomous operators would significantly increase this system's versatility.

References

[1] W. Jacak and J.W. Rozenblit, "Automatic Simulation of a Robot Program for a Sequential Manufacturing Process" *Robotica*, Vol. 10, pp. 45-56, 1992.

[2] B.P. Zeigler, *Object-Oriented Simulation with Hierarchical, Modular Models,* Academic Press, London, 1990.

[3] B.P. Zeigler, *Multifacetted Modelling and Discrete Event Simulation* Academic Press, London, 1984.

[4] B. Faverjon, "Object Level Programming of Industrial Robots" *IEEE Int. Conf. on Robotics and Automation*, Vol. 2, pp. 1406-1411, 1986.

[5] R. Speed, "Off-line Programming for Industrial Robots" *Proc. of ISIR 87*, pp. 2110-2123, 1987.

[6] N.J. Nilsson, *Principles of Artificial Inteligence*, Tioga, Palo Alto, CA, 1980.

[7] H.A. ElMaraghy and J.M. Rondeau, "Automated Planning and Programming Environments for Robots" *Robotica*, vol. 10, pp. 75-82, 1992.

Supervising Manufacturing System Operation by DEVS-Based Intelligent Control

H. Praehofer, G. Jahn, W. Jacak, G. Haider

Institute of Systems Science
Johannes Kepler University Linz
A-4040 Linz, Austria

Abstract

The purpose of a flexible manufacturing system (FMS) is to perform a series of well defined operations on a family of similar parts. The operations (e.g. milling or drilling) are realized by machines which are serviced by robots. A cell controller coordinates the flow of parts through the cell. Monitoring, i.e., to watch the system behavior during its operation and detect possible anomalies, is an important task of an intelligent manufacturing controller. This paper present an approach for manufacturing system monitoring which has its foundation in the DEVS-based intelligent control paradigm. The work is part of a bigger research project whose objective is to develop techniques for automatic synthesis of intelligent flexible manufacturing system controllers. Important additional parts of the intelligent control system will provide fault diagnosis and self repair capabilities.

1 Introduction

The purpose of a flexible manufacturing system (FMS) is to perform a series of well defined operations on a family of similar parts. The operations (e.g. milling or drilling) are realized by machines which are serviced by robots. In addition, stores within the cell can hold parts temporarily.

A cell controller coordinates the flow of parts through the cell and supervises the underlying machines and robots. The control system of such a workcell can be divided into three levels [5]:

- The *organization* level accepts and interprets related feedback from the lower levels and defines the strategies for task sequencing.
- The *coordination* level realizes the strategies given by the organization level. It defines part routing in the logical and geometric aspects and coordinates the activities of workstations and robots. Additionally, it supervises the lower control levels [5].
- The *execution* level consists of the actual device controllers which execute the programs generated by the coordinator.

In [1, 3, 4, 5, 2] an approach for automatic synthesis of intelligent hierarchical controller for flexible manufacturing cells is developed. The approach uses *off-line* planning and *on-line* real-time monitoring and control. For off-line planning two types of simulation are employed. *Continuous simulation* is used for motion and trajectory planning. A *discrete event hierarchical DEVS simulation model* is used for verification and testing of different variants of task realizations obtained from a route planner.

On-line monitoring and control is accomplished employing an internal DEVS model of the workcell and real-time on-line simulation. In [5] a real-time discrete event simulator is described which is used for predictions of some motion commands of robots and for monitoring of process flow. In [2] a fuzzy rule based decision system is applied to create an organizer for finding optimal strategies for task sequencing.

In this paper we focus on the monitoring task of the controller. We present a monitoring system which slightly differs from the approach taken in [5] and which uses DEVS-based discrete event control as introduced by Zeigler [11, 12]. The approach taken here minimizes the overhead for controller synchronization and allows one to make precise decisions about system malfunctions.

In the following, we first shortly review the DEVS-based discrete event control paradigm, then we describe the DEVS-based monitoring system in detail, we discuss the simulation model set up for testing and validation of the concepts, and we conclude with a summary and outlook on future research.

0-8186-6440-1/94 $04.00 © 1994 IEEE

2 DEVS-Based Control Reviewed

Conventional discrete event control schemes [9, 10, 7] are based on so-called *logical model* of discrete event systems which do not use the concept of *holding time*, i.e., the time duration the system stays in a particular logical state. In contrast to that, Zeigler introduced the *event-based intelligent control* paradigm [11, 12]. An event-based controller bases its control actions not only at the sensor events coming from the system under control but also on the time of the sensor reactions. In event-based intelligent control the controller exerts control commands to the real system and expects to receive confirming responses from the system within a definite time window - not too early and not too late. A set of threshold like sensors are used in the real system to report process evolution. As long as the sensor responses received are in correspondence with the expected ones, i.e., the correct sensor responses are received in the appropriate *time window*, the process run is regarded to be correct. The information of the correct sensor responses is codified in a model internal to the event-based controller. The internal model is an abstraction of the real system which is based on a distinction between normal and abnormal system behavior. In most applications, parameter variations in finite intervals may be tolerable and are regarded as normal. The event-based control model is derived by considering all system behaviors for all parameter variations which are accepted as normal.

3 DEVS-based Controller of FMS

3.1 Robotized FMS Cell

Flexible manufacturing systems under consideration are constructed as depicted in Figure 1. They consist of a set of NC-programmable machines and production stores, so-called magazines, connected by a flexible material handling system, typically a robot or an automated guided vehicle, and controlled by a computer system. Machines have a number of buffer places where robots can place workpieces and from where workpieces are loaded automatically into the machine for operation. Buffer places and magazine places have threshold sensors which signal if a workpiece is currently stored in the buffer place. Except one sensor which provides detailed information about the workpiece upon entrance into the system, these threshold sensors are the only information the controller receives from the real system. The controller is mainly responsible for control of the material handling system, the machines are equipped with their own local control system.

The *technological task* which is realized by a flexible manufacturing cell defines the sequence of operations which can be performed on the various workpieces. The technological task is crucial for the function of the control system. It consists of a set of operations together with partial ordering of the operations and a relation of device or storage assignments. A detailed formal description of the cell and technological task realization is given in [4, 5, 6].

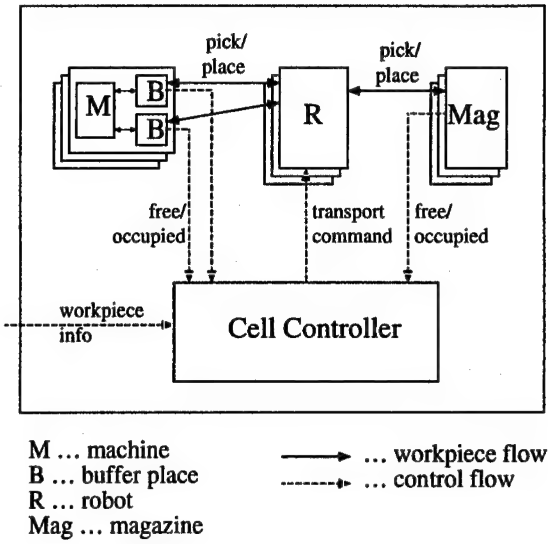


Figure 1: Flexible manufacturing cell.

3.2 Controller of FMS Cell

The controller for a manufacturing cell has to fulfill the following tasks:

- *Generation of transport commands:* It has to generate the transport commands for the robots based on technological tasks, information about workpieces waiting to be transported, and strategies for task sequencing.
- *Monitoring:* It has to monitor if the various tasks are fulfilled correctly, i.e., if transport of workpieces by the robots and the processing of workpieces by machines are finished successfully.
- *Diagnosis:* In case incorrect system behavior is detected, the controller should find its cause.
- *Repair:* After diagnosis the controller has to determine how the malfunction can be repaired. In particular, it should update its own internal state to a state which corresponds to the system's state and which allows the controller to continue.

In [5] a method is presented how generation of transport commands can be accomplished so that tasks are

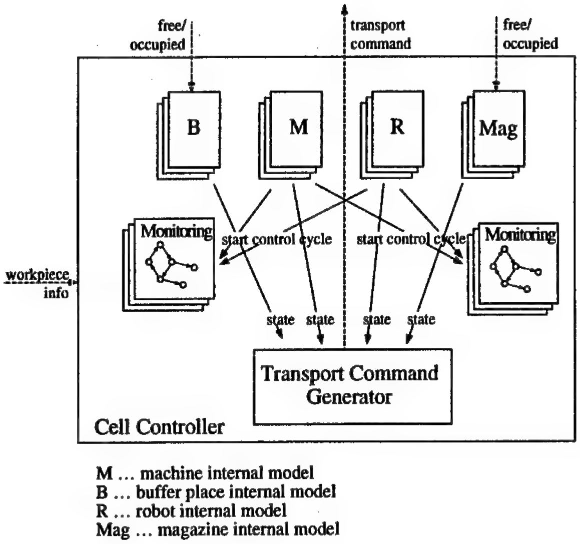


Figure 2: Components of a DEVS-based controller.

realized, deadlock is avoided and systems performance is optimized for various criteria. In this paper we concentrate on the monitoring task of the controller and show how monitoring can be accomplished employing the DEVS-based control paradigm reviewed above.

3.3 Monitoring FMS Operations

Figure 2 shows the interface and the various parts of a DEVS-based controller for monitoring. Inputs to the controller are the threshold sensors which, for each place, signal if the place is free or occupied by a workpiece. Additionally, it has one extra input through which the controller gets detailed information about a workpiece when the workpiece enters the system. In particular, this contains information about the sequence of operations which have to be carried out. Outputs from the controller are the transport commands for the robots.

Most important for the controller is information about the current state of the real system. This is represented by internal passive models for the active components, i.e., robots and machines, and for the passive places, i.e., machine buffers and magazines. Based on information provided by the threshold sensor, the controller tries to keep the state of the internal models in correspondence to the state of the real system components. Therefore, the internal models are passive reacting to the inputs to the controller and to the transport commands generated.

Additionally to the internal passive models, the controller employs active *monitoring DEVS models* for every place. These monitoring models are used to check if transport and machining operations for workpieces are fulfilled in the correct time. This is accomplished based on the DEVS-based control paradigm as reviewed above. With each transport and machining operation, the monitoring DEVS is activated to run through a *monitoring cycle.* Information about the timing constraints of various robot transports and various machining operations has to be provided by the user and is represented as tabular information in the internal models of robots and machines.

In the following the various parts are described in detail.

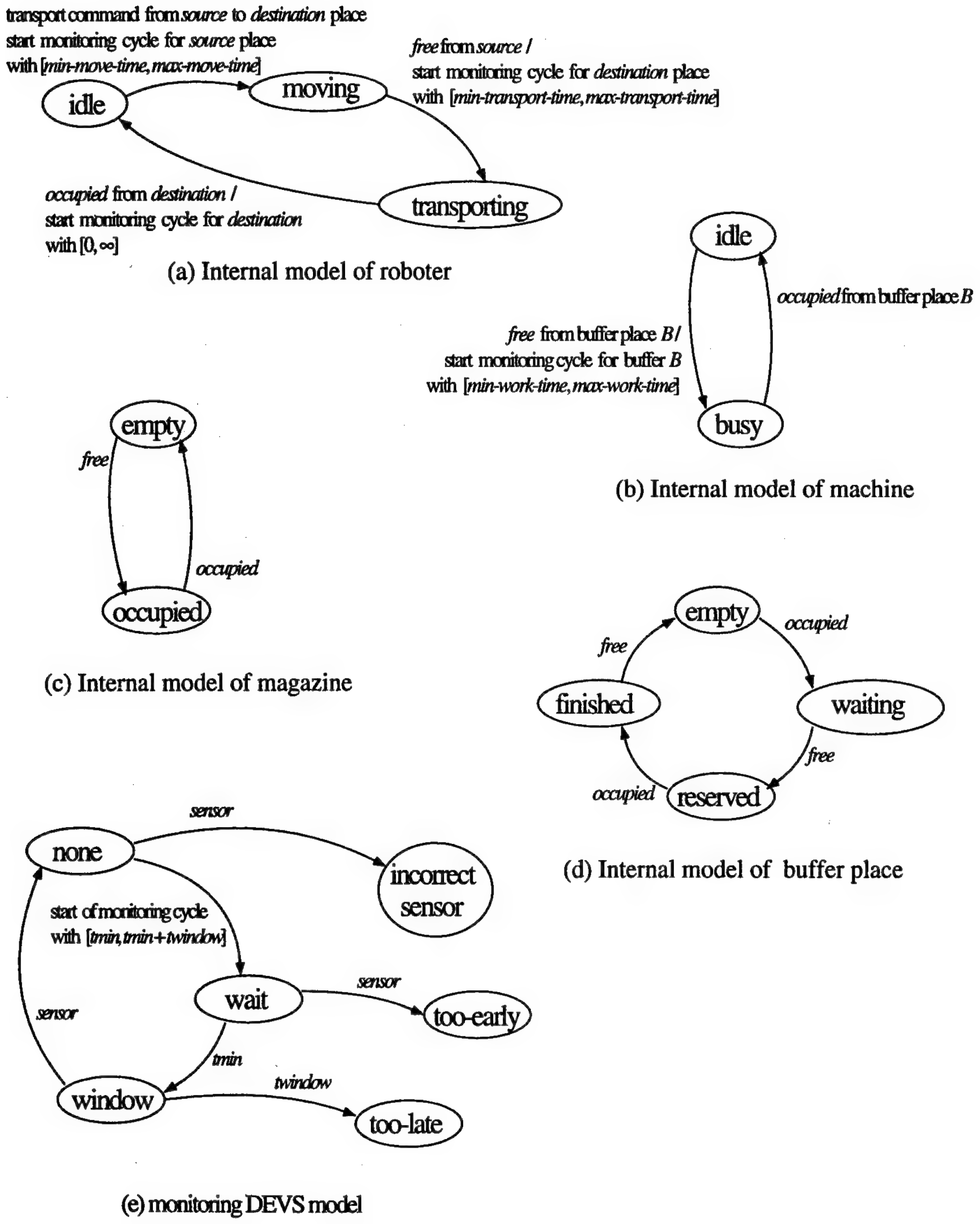
Transport command generator

The transport command generator examines the information about workpieces stored in the internal models of the places and selects one to be transported next. The selection process has to be done to meet various objectives. First, deadlocks in the manufacturing cell has to be avoided. Then, throughput should be maximized and turnaround time should be minimized. From the organizing level different strategies for scheduling operations are provided, e.g. just in time, maximum waiting time, minimum setup, pull strategy, first free buffer, or minimum transfer cost [2]. A transport command generator which accomplish this is described in detail in [5, 6].

Robot internal model

The internal model of the robot is depicted in figure 3(a). In correspondence with the real robot, it traverses different phases which show the current state in fulfillment of the robot's transport task. Upon the output of a new transport command by the transport command generator, the internal model transits from phase *idle* to phase *moving.* This represents the *source* phase where the real robot moves to the place to pick the workpiece. This phase terminates when the workpiece is picked up and the sensor of the place switches to *free.* Upon entering the phase *moving* a monitoring cycle is started for the source place which checks if this sensor reaction caused by picking up the workpiece occurs in the right time window.

After phase *moving*, the phase *transporting* follows. This phase terminates when the sensor of the destination place switches to *occupied.* The occurrence of the sensor reaction in the correct time window is checked by a monitoring cycle for the destination place which is started when entering phase *transporting.* The sensor reaction in the destination place makes the internal robot model to transit to its initial state *idle* which signals the transport command generator that the robot is ready to accept the next transport command. With that transition we start a control cycle in the destination place with a time window of 0 to infinity which


transport command from *source* to *destination* place
start monitoring cycle for *source* place
with [*min-move-time, max-move-time*]
idle
moving
free from *source* /
start monitoring cycle for *destination* place
with [*min-transport-time, max-transport-time*]
transporting
occupied from *destination* /
start monitoring cycle for *destination*
with [0, ∞]
idle
occupied from buffer place *B*
free from buffer place *B* /
start monitoring cycle for buffer *B*
with [*min-work-time, max-work-time*]
busy
empty
free
occupied
occupied
empty
occupied
free
finished
waiting
free
occupied
reserved
none
sensor
incorrect sensor
start of monitoring cycle
with [*tmin, tmin+twindow*]
wait
sensor
too-early
sensor
tmin
window
twindow
too-late

(a) Internal model of roboter

(b) Internal model of machine

(c) Internal model of magazine

(d) Internal model of buffer place

(e) monitoring DEVS model

Figure 3: Transition diagrams of internal models.

means that the machine can load the workpiece for processing any time it wants.

Machine internal model

Figure 3(b) shows the simple phase diagram of the machines. Machine control is not done by the main controller but is local in the particular machines. Any sensor reaction in a buffer place of a machine which does not correspond with a current transport command of a robots is considered to be caused by the machine or is erroneous. As soon as a workpiece is loaded from a buffer place into the machine, which is signaled by the sensor switching to *free*, the internal model of the machine transits to *busy*. A control cycle is started to check that processing the workpiece is finished in the correct time window. When the workpiece is put back to the place, the machine internal model transits to *idle* again.

Buffer place and magazine internal model

The internal model of magazines is shown in figure 3(c). It strictly goes hand in hand with the current state of the sensor of the magazine. The internal model also holds detailed information about the workpiece it holds, in particular, it knows the remaining operations which have to be performed.

Because robots as well as machines have access to the buffer places of machines, the internal models of buffer places are more complicated (Fig. 3(d)). When a new workpiece is placed on the buffer by the robot, the internal model transits to *waiting*. When the sensor goes to *free* again the workpiece is taken by the machine to be processed and the internal model goes to *reserved* meaning that the place is reserved for the workpiece currently processed. When the workpiece is put back after the operation has been performed, the internal model transits to *finished* and there waits that the robot gets the workpiece and transports it to the next destination. Upon that, the internal model transits to its original state *empty*.

Monitoring DEVS model

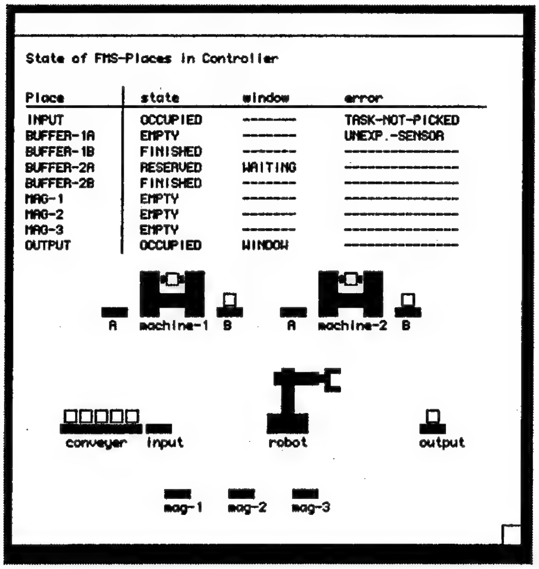
For each transition in the internal models of places caused by a sensor reaction, a monitoring cycle of the respective monitoring DEVS exists. The monitoring cycles are started with transitions of robot and machine internal models (see Fig. 3(a) and (b)). Figure 3(e) shows the phase diagram of the monitoring DEVS models associated with the places. Initially, the model is in phase *none* which means that no sensor reaction is expected. A sensor reaction in this phase means that an error *incorrect sensor reaction* is detected. When the monitoring cycle is started with a time window $[tmin, tmin + twindow]$ the model transits to phase *wait* to wait the time it takes until the sensor values are expected. This time span is given by *tmin*, the minimum time of the time window. An receipt of a sensor in that phase means that the sensor reaction is too early which causes the controller to transit to error phase *too-early*. When *tmin* time units have elapsed without any sensor response, the time window begins where the sensors reactions are expected and the model will transit to phase *window*. If the sensor is received in that phase, the process run is recognized to be correct and the monitoring cycle is terminated. The time span for phase *window* is given by the length of the time window *twindow*. A *too-late* error occurs if the time for phase *window* elapses without any sensor reaction. The error outputs generated in the respective error phases can be used by a diagnosing unit to decide on the cause of the error.

4 Simulation Model of FMS

A simulation system has been developed to validate the control scheme. It uses to STIMS modeling and simulation environment [8]. STIMS is a new simulation environment which is implemented in CommonLisp/CLOS and which allows modular hierarchical DEVS and DEVS-based combined discrete/continuous modeling and simulation. The FMS simulation system will take as input a definition of the configuration of the FMS and then will synthesize the simulation model automatically. The definition of the configuration has to contain information of the set of machines, robots, magazines together with information which machine buffers and which magazines are serviced by which robots, an assignment of operations to machines, information of the service times for the different operations, and time windows for the different operations. To test the DEVS-based controllers, malfunctions, like breakdown of sensors, breakdown of machines, losses of pieces by robots etc. are built into the models of the real system. The behavior of the model can be studied by a simple animation system showing movements of parts, malfunctions and the reactions of the DEVS-based controllers. Figure 4 shows a snapshot of animation of a sample manufacturing system. On the right the states of the internal models are listed.

5 Summary and Outlook

The paper presented an approach for monitoring operations of flexible manufacturing systems which is based on the DEVS-based intelligent control paradigm. Based on abstract models of real system components, the monitoring system is able to detect anomalies in the system behavior.



State of FMS-Places in Controller

Place	state	window	error
INPUT	OCCUPIED	--------	TASK-NOT-PICKED
BUFFER-1A	EMPTY	--------	UNEXP.-SENSOR
BUFFER-1B	FINISHED	--------	---------------
BUFFER-2A	RESERVED	WAITING	---------------
BUFFER-2B	FINISHED	--------	---------------
MAG-1	EMPTY	--------	---------------
MAG-2	EMPTY	--------	---------------
MAG-3	EMPTY	--------	---------------
OUTPUT	OCCUPIED	WINDOW	---------------

Figure 4: Animation of FMS monitoring.

The work is part of a bigger research project whose objective is the development of techniques for automatic synthesis of intelligent flexible manufacturing system controllers. Solutions for several subtasks like off-line task and route planning [3], on-line operation scheduling [5, 6], on-line strategy determination for task sequencing [2], and system operation monitoring have been provided. Fault diagnosis and self repair capabilities in flexible manufacturing is the topic of an ongoing research project.

References

[1] W. Jacak. A discrete kinematic of robot in the cartesian space. *IEEE Transactions on Robotics and Automation*, 5(4):435–446, 1989.

[2] W. Jacak. Fuzzy rule based control of intelligent robotic workcells. In *Cybernetics and Systems*, pages 91–100, Vienna, April 1994. World Scientific.

[3] W. Jacak and J. Rozenblit. Automatic robot programming. *Robotica*, 10(3), 1992.

[4] W. Jacak and J. Rozenblit. Cast tools for intelligent control of manufacturing automation. *Lecture Notes in Computer Science*, 763:203–219, 1993.

[5] W. Jacak and J. Rozenblit. Virtual process design techniques for intelligent manufacturing. In *Proc. of AI, Simulation and Planning in High-Autonomy Systems*, pages 192–198, Tucson AZ, Sept 1993. IEEE/CS Press.

[6] W. Jacak and J. Rozenblit. Model-based workcell planning and control. *IEEE Transactions on Robotics and Automation*, in print.

[7] M.D. Lemmon and P.J. Anstaklis. Hybrid systems and intelligent control. In *Proc. of the 1993 IEEE International Symposium on Intelligent Control*, pages 174–179, Chicago, IL, 1993.

[8] H. Praehofer, F. Auernig, and G. Reisinger. An environment for DEVS-based multiformalims simulation in Common Lisp / CLOS. *Discrete Event Dynamic Systems: Theory and Application*, 3(2):119–149, 1993.

[9] P.J.G. Ramadge and W.M. Wohnham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[10] J. A. Stiver and P.J. Anstaklis. Extracting discrete event models from hybrid control systems. In *Proc. of the 1993 IEEE International Symposium on Intelligent Control*, pages 298–301, Chicago, IL, 1993.

[11] B. P. Zeigler. DEVS representation of dynamical systems: Event-based intelligent control. *Proceedings of the IEEE*, 77(1):72–80, 1989.

[12] B. P. Zeigler. *Object-Oriented Simulation with Hierarchical, Modular Models.* Academic Press, London, 1990.

Session 2D:

DEVS Formalism: Discrete Event Systems

The DEVS framework for Discrete Event Systems Control *

Hae Sang Song and Tag Gon Kim
Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
Taejon 305-701, Korea
tkim@ee.kaist.ac.kr

Abstract

This paper proposes a new methodology for analysis of discrete event systems and design of discrete event systems controllers. The methodology is based on the sound semantics for specification of discrete event systems called the DEVS formalism. It introduces concepts of inverse DEVS and defines controllability of discrete event systems expressed in the DEVS formalism. These two concepts, inverse DEVS and controllability of discrete event systems, play important roles in designing a discrete event controller. An example for appreciating the concepts is presented.

1 Introduction

Discrete event systems (DES) have taken a more important part in managing the contemporary world, most of which are man-made systems such as multicomputer systems, communication networks, traffic systems and manufacturing systems. In such a DES, computation is done by interactions between components to achieve a given goal. Such a goal can be an operation range of system behavior or optimization of system performance. An operation range of a DES can be specified by a state trajectory which is piecewise constant in time function. In the control system's viewpoint, a DES can be divided into two subsystems, plant and controller. Thus, the DES control problem is as follows. Given a DES plant, design a DES controller to meet specified objectives.

Compared to numerous efforts to develop formalisms to specify the discrete event systems, there are a few researches on design of a discrete event controller (DEC). [1] is the first work in the field, where a DES plant is specified based on automata theory and a discrete event controller is designed based on language theory. The essential feature is that a controller has the desired behavior of a plant with respect to objectives. The controller supervises output events occurring in the plant and produce required control outputs to the plant.

This paper proposes a framework for specification and design of discrete event systems control based on the DEVS (discrete event systems specification) formalism [2]. Unlike the methodology based on automata theory [1], we base DES controller design on the system-theoretic DEVS formalism. DEVS, in nature, has more information than that of automata in specification, especially timing information. It also distinguishes state transition into two different ones, i.e., internal and external transitions and discriminates input events from output events. Our approach inherits diverse advantages of the DEVS formalism.

2 Supervisory control of discrete event systems

Supervisory control [3] is a kind of feedback control that supervises the behavior of a discrete event plant to control the plant in desired objectives. It is assumed that the behavior of the plant is known. Thus the controller can deduce the current behavior of the plant from observed output events. Then it compares the behavior with desired one and generates appropriate next control outputs. A design problem of a discrete event controller is as follows: given a discrete event system with known dynamics and control objectives, design a discrete event controller that satisfies the control objectives.

*ISBN 0-8186-6440-1.

0-8186-6440-1/94 $04.00 © 1994 IEEE

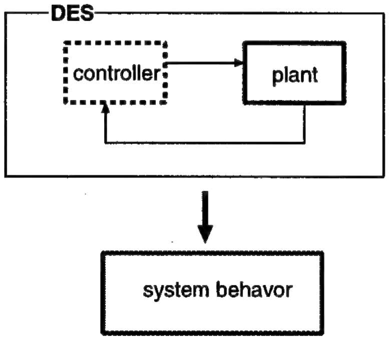


Figure 1: Supervisory control of a discrete event system

Fig. 1 shows concepts of supervisory control. To design a discrete event controller, we have to specify 1) dynamics of a plant 2) controlled behavior of the plant that satisfies the control objectives. In this paper, we specify plant behavior based on the DEVS formalism. We call it a plant DEVS. From informal control objectives and the plant DEVS, we obtain a desired state trajectory that satisfies the control objectives. A discrete event controller would be obtained by transformation of the desired state trajectory into a DEVS model. Connecting the plant with the designed controller through proper interfaces, called ports, results in an overall control system satisfying the desired system behavior. We will describe concepts and design steps in more detail in the following sections.

3 The DEVS formalism

This section briefly reviews the DEVS formalism and introduces a graphical notations representing the DEVS formalism.

3.1 Review of the DEVS formalism

The DEVS formalism specifies discrete event models in a hierarchical, modular form. Formally, an atomic model M is specified by a 7-tuple :

$$M =< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$$

where

X : input events set;
S : states set
Y : output events set;
$\delta_{ext} : Q \times X \rightarrow S$: external transition function;
$\delta_{int} : S \rightarrow S$: internal transition function;
$Q = \{(s,e) | s \in S, 0 \leq e \leq ta(s)\}$: total states;
$\lambda : S \rightarrow Y$: output function;
$ta : S \rightarrow Real$: time advance function.

The four elements in the 7-tuple, namely, δ_{int}, δ_{ext}, λ, ta are called characteristic functions, and S is set of state variables, $X(Y)$ is set of input(output) events. An atomic model represents a corresponding discrete event process and connections between processes is represented by a coupled model DN

$$DN =< X, Y, M, EIC, EOC, IC, SELECT >$$

where

X : input events set;
Y : output events set;
M : DEVS components set;
$EIC \subseteq DN.IN \times M.IN$: external input coupling relation;
$EOC \subseteq M.OUT \times DN.OUT$: external output coupling relation;
$IC \subseteq M.OUT \times M.IN$: internal coupling relation;
$SELECT : 2^M - \emptyset \rightarrow M$: tie-breaking selector.

The three elements, EIC, EOC, IC means the connections between set of models M and input, output ports X, Y. $SELECT$ function acts as a tie-breaking selector.

3.2 Dynamics of a DEVS model

The dynamics of a discrete event systems specified in the DEVS formalism can be interpreted by the abstract simulator in [2]. We mention it briefly here and define some terminologies to be used in the following sections. The dynamics of an atomic DEVS is determined by a sequence of internal or external transitions. An internal transition is spontaneous after fulfilling its current activity and an external transition is caused by an external event.

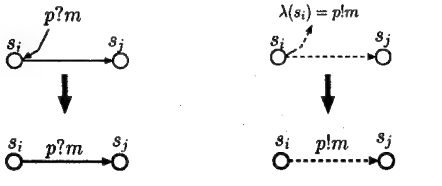
An atomic DEVS M_i spontaneously changes its current activity(or state) s_i to $\delta_{int}(s_i)$ after completing the activity at the predefined sojourn time $ta(s_i)$. Just before the transition, M_i produces an output event $\lambda(s_i)$. The stimulus that makes this transition to occur is generated at which an activity has just completed. We call the stimulus a * event. It's not visible to us and other DEVS models, so we call it a *hidden event*. This transition, $* \rightarrow \lambda(s_i) \rightarrow \delta_{int}(s_i)$, is called an *internal transition*. From the above statement, we can define *internal transition relation* for a DEVS M_r as follows: $T_{r,int} = \{(s_i, p!m, s_j) | (s_i, s_j) \in \delta_{r,int},\ p!m = \lambda_r(s_i) \in Y_r, s_i \in S_r\}$.

An output event $p_i!m_i$ resulting from an internal transition of M_i is converted into an input event $p?m$ of another DEVS M_j connected with M_i through the port p. If M_j is ready to receive the input event $p?m$ at the current state s_j, it changes its state into $\delta_{ext}(s_j, p?m)$. We call the transition an *external transition.* We define *external transition relation* of a DEVS M_r as follows: $T_{r,ext} = \{(s_i, p?m, s_j) | (s_i, p?m, s_j) \in \delta_{r,ext}, s_i \in S_r\}$.

We can see that an external transition of a DEVS is caused by an output event resulting from an internal transition of another DEVS stimulated by the * event. That is, with the * event, two DEVSs are changing their states at the same time. This phenomenon is *concurrency.* We deal with concurrency later in more detail. A pair of input and output events, $p_i!m_i$, $p_j?m_j$, that make concurrency satisfies the following properties: $dom(p) = dom(p_i) = dom(p_j)$ and $m = m_i = m_j$, where $dom(p)$ is the domain of port p that messages can reside. We call the pair a *dual event.*

3.3 Graphical notations

An atomic model representing a process is enclosed by a box with input and output ports in the wall. Such Ports are entrances or exits for messages on its own to represent X and Y in DEVS. Combinations of ports and messages by ?(input) and !(output) represent input events on X and output events on Y, respectively. Each state variable in the state set S of DEVS is represented by a small box in the atomic model and has its name in it. The behavioral description of an atomic model is represented by an *activity transition diagram* or an state transition diagram, which consists of nodes and two-colored edges. Each node represents an activity or a state, dotted arc denotes an internal transition and solid arc an external transition. In Fig. 2, (a) represents an external transition and (b) denotes an internal transition consisting of $\lambda(s) \rightarrow \delta_{int}$. An output event is specified on a dotted line by an output port followed by a message name with output operator '!'. For example, an output event $out!m$ means that a message m is output at the port out. Similarly an input event is specified on a solid line by an input port followed by a message name with input operator '?'. An input event $in?m$ means that a message m is input at the input port in. The change of values of state variables are enclosed by '[' and ']'. If it is on the dotted (solid) line, it represents a state transition specified by the internal (external) transition function δ_{int} (δ_{ext}) of DEVS, respectively. Optionally, transition condition g can be specified after input or output event with a seperator notation @. It is natural that a time advance in a state be attached to the state node because it represents a sojourn time to fulfill its activity. An empty output event is denoted by $\emptyset$.



(a) $\delta_{ext}(s_i, p?m) = s_j$

(b) $\lambda(s_i) = p!m \rightarrow \delta_{int}(s_i) = s_j$

Figure 2: graphical notation: (a) external transition (b) internal transition

4 Concurrency

Consider two atomic models M_i and M_j, $M_i = < X_i, S_i, Y_i, \delta_{i,int}, \delta_{i,ext}, \lambda_i, ta_i >$, $M_j =< X_j, S_j, Y_j, \delta_{j,int}, \delta_{j,ext}, \lambda_j, ta_j >$. Assume that $M_i || M_j$ be a composition of M_i and M_j. Let state transitions of M_i, M_j be $(q_i, a, r_i) \in (T_i - T_j) = T_{i,int} \cup T_{i,ext}$, $(q_j, b, r_j) \in (T_j - T_i) = T_{j,int} \cup T_{j,ext}$. Then state transition relation T of the composition $M_i || M_j$ has the following transition rules:

(1) if $a = p!m$ and $b \neq p?m$ then
$\rightarrow ((q_i, q_j), p!m, (r_i, q_j)) \in T$, for all $q_j \in S_j$

(2) if $b = p!m$ and $a \neq p?m$ then
$\rightarrow ((q_i, q_j), p!m, (q_i, r_j)) \in T$, for all $q_i \in S_i$

(3) if ($a = p!m$ and $b = p?m$) or ($b = p!m$ and $a = p?m$) then
$\rightarrow ((q_i, q_j), p\#m, (q_i, r_j)) \in T$

(4) No other transition in T

(3) means if a, b is dual event and connected, then both are transiting concurrently. That is, if an output event resulting from an internal transition of one model becomes an input event of the other waiting for it, then the two models changes their states concurrently. This is called concurrency and the event communicating the message is called *concurrent event,* denoted by $p\#m$. (1) and (2) represent the cases even if an output event of one model is produced but the other one is not waiting for the event in its current state. In this case, the output event takes no effect on

the other model and disappeared. We call the disappeared event a *dangling event.*

The concurrency rules described above will be used to analyze the dynamics of a system consisting of several subsystems.

5 Controllability and inverse DEVS

Controllability of a discrete event system specified in DEVS is exploited to check if a discrete event controller satisfying a desired state trajectory exists or not. If controllable, concepts of inverse DEVS would transform the desired state trajectory into the behavior of a discrete event controller. In this section, we define these two concepts.

5.1 Controllable DEVS

Let T be the global state trajectory of a discrete event plant $M =< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$. Let $ST \subseteq T$ be a desired state trajectory.

Definition 1 (strong controllability) *A desired state trajectory ST is said to be* strongly controllable *if the following condition is hold. For all s_i on ST, if $\exists\ s_j = \delta_{int}(s_i)$, then s_j is on ST. A discrete event plant is said to be strongly controllable if $\forall\ ST \subseteq T$, ST is strongly controllable.*

Definition 2 (weak controllability) *A desired state trajectory ST is said to be* weakly controllable *if the following condition is hold. For all s_i on ST, if $\exists\ s_j = \delta_{int}(s_i)$ not on ST, then $\exists$ at least one $s_k = \delta_{ext}(s_i, e, p?m)$ such that s_k is on ST and $e < ta(s_i)$. A discrete event plant is said to be weakly controllable if $\forall\ ST \subseteq T$, ST is weakly controllable.*

Note that controllability in Def. 1 does not depend on time, but Def. 2 is time-dependent.

5.2 Inverse DEVS

Consider two DEVS M_i, M_j operating concurrently. If $M_i||M_j$ is running with transitions caused only by concurrent events (see Section 4), then we call $M_i(M_j)$ an inverse DEVS of $M_j(M_i)$, respectively.

Definition 3 (inverse DEVS) *$M_i(M_j)$ is said to be an* inverse DEVS *of $M_j(M_i)$ iff the following properties are hold:*

(i) State set morphism

$$S_i \rightarrow S_j$$

(ii) Dual I/O events set

$$X_i = \{p?m|p!m \in Y_j\}$$
$$Y_i = \{p!m|p?m \in X_j\}$$

(iii) Dual transition relation

$$T_{i,int} = \{(q, p!m, r)|(q, p?m, r) \in T_{j,ext}\}$$
$$T_{i,ext} = \{(q, p?m, r)|(q, p!m, r) \in T_{j,int}\}$$

(i) states that there is one-to-one correspondence between states of two DEVSs. (ii) indicates that input events of one would be converted into output events of the other, and vice versa. (iii) denotes that an internal transition of one DEVS would be changed to an external transition of the other.

A transformation from an atomic DEVS M_i to M_j satisfying the above properties is called an *inverse DEVS transformation.* The concepts of inverse DEVS is used to obtain a discrete event controller from a desired controlled state trajectory.

6 Design methodology for DEC

A methodology for design of a DES controller bases on the DEVS formalism and the concepts of inverse DEVS. Fig. 3 shows the steps for design of a discrete event controller. First, a plant is specified in the DEVS formalism, which consists of one or more atomic models that are connected each other. We can obtain a global state space at this step, from which we shall extract a controlled state trajectory satisfying given objectives. We call it a desired state trajectory of the plant. The next step is to check the controllability of the desired state trajectory by the controllability definition defined in the paper. And the supervisor DEVS transformation described above is used to directly transform the desired state trajectory to a DES controller. After the transformation we relabel states of the controller after suitable state reduction. Finally we can obtain a controller. Note that if a desired state trajectory of a plant is given, then we can directly obtain a corresponding DES controller from it using the inverse DEVS transformation.

7 A simple example: control of water supply system

We will clarify the approach described previous sections by a simple example of a water supply system.

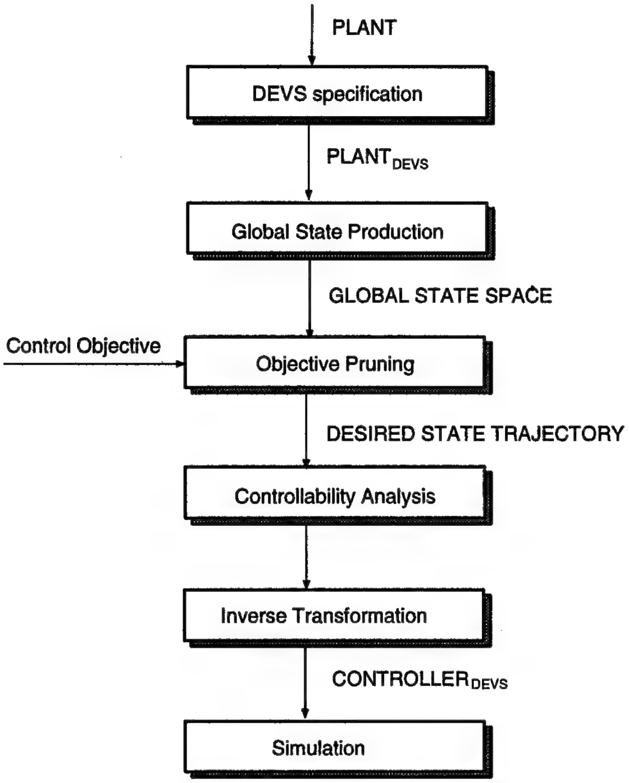


Figure 3: Design steps for a discrete event controller

Following subsections show how each step in Fig. 3 performs through the example. We use the graphical notations rather than set-theoretical notations, for graphical notations show more clear model descriptions.

7.1 Problem description

An informal problem description is a starting point for the design of a controller for the water supply system. In Fig. 4, a water supply plant consists of two subsystems, a water pump and a water tank. In the water tank, there are two water level sensors (low, high) and a water pipe through which water flows in. The water pump takes electric energy from outside world, which can be controlled by a ON/OFF switch. It fills the water tank through the water pipe.

The purpose of the system is to keep the water level between the low and high sensors.

7.2 DEVS model of the plant

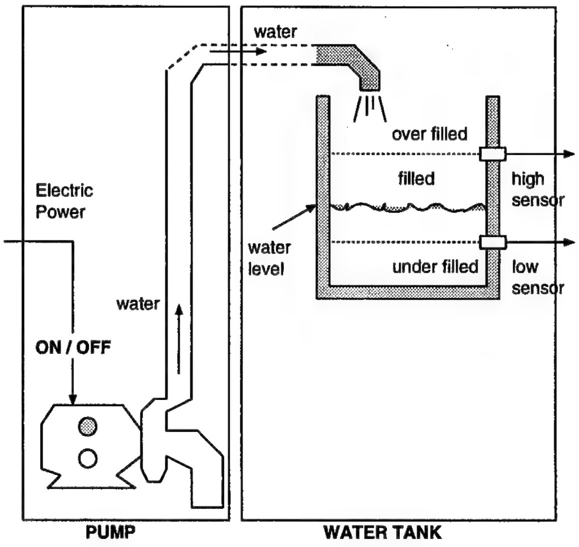


Figure 4: Water pump plant

Specification of a plant in the DEVS formalism is the first step to design a discrete event controller. Input and output events, state variables, output function, internal and external transition functions and time advance function should be identified.

To specify a plant in the view of a discrete event system, we have to identify events occuring in the plant. The events set of the water pump consists of ON/OFF switching events of the motor, and events that water starts and stops to flow out through the water pipe. The events set of the water tank consists of the low sensor ON/OFF, the high sensor ON/OFF, and events that water starts and stops flowing in through the water pipe.

Fig. 5 shows a DEVS model of the plant. If the water pump is turned on ("pw"?ON), it starts pumping water through the water pipe ("w"!WSTRT). If it is turned down ("pw"?OFF), it stops pumping and the flow of water will be stopped ("w"!STOP). Initially, the water tank changes INIT state into BS, FS, VS according to the initial water level producing appropriate sensor signals to outside world. The letter B means under-filled, F filled, V over-filled and the second letter S means that the in-flow of water is currently stopped. While the water level lowers, corresponding level sensor signals would be generated. With water flowing in, the state is changed into BF, FF, VF generating proper sensor signals. If the in-flow is continued even though the water level reaches far over the high sensor, the water tank eventually overflows (TOP). In the opposite case, the water level would reach to the bottom (BOT).

The two subsystems, the water pump and the water tank, are connected through the pipe ("w"). The dynamics of the composite subsystem can be obtained

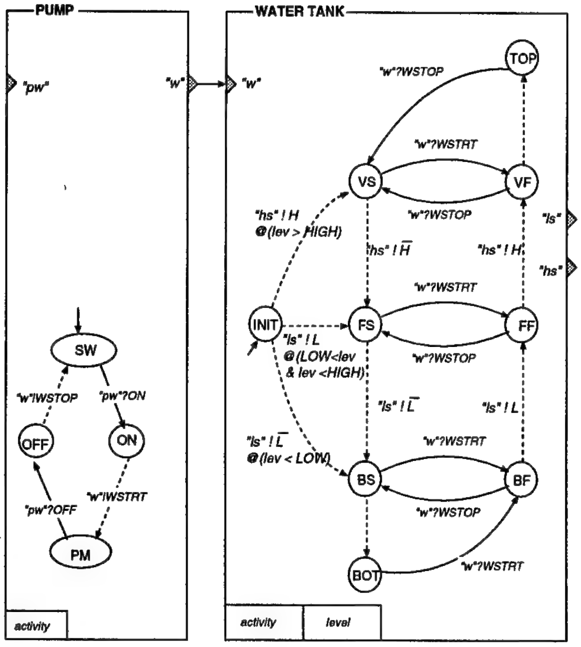


Figure 5: Specification of the water pump plant

by concurrency analysis.

7.3 Controlled state trajectory

The next step is to get a global state transition diagram (GSTD) from the plant specification. It may be obtained by a Cartesian product of state diagrams of each subsystem. Alternatively, if we use the concurrency analysis, we can obtain far less number of states. In the case of no connection, there would be $4 \times 9 = 36$ states in the example plant. However, there is one connection("w") and we obtained a GSTD of the plant with only 17 states by the concurrency analysis.

Meanwhile, the informal description of control objectives is to be re-described based on the DEVS model of the plant. Let a controlled state trajectory be K, and s on GSTD be $(P.s, W.s)$, where $P.s$ denotes a state of the water pump and $W.s$ a state of the water tank. Then K should satisfy the following conditions:

1. If $\exists$ (x, BS), then eventually (SW, VS), where x means *don't care.*
2. $\exists$ no s such that (x,BOT) or (x,TOP).
3. hysteresis: $\exists$ no transition between (x,FF) and (x,FS)

Now we can extract a controlled state trajectory from GSTD and the control objectives. Fig. 6 is the

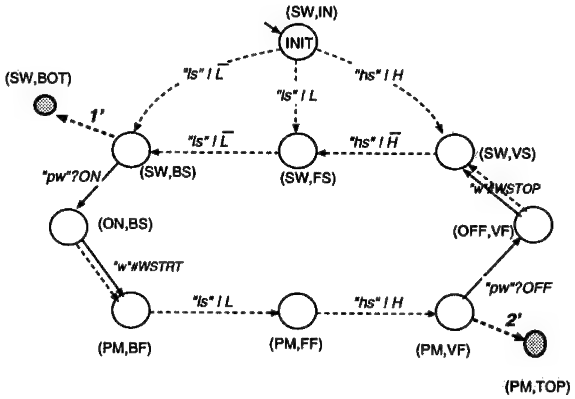


Figure 6: desired state trajectory of the plant

desired state trajectory (In the figure, we exclude transition 1' and 2' from the desired state trajectory, which were added for convenience to explain controllability.). Initially, the state is changed into (SW,BS) after a few internal transitions. Then the motor should be ON, which is followed by a concurrent event "w"#WSTRT. If the water level reaches to the over-filled region, the motor should be OFF. Then, water stops flowing in. By internal transitions, the water level is to be lowered again, and then the motor should be ON, and so on. This state trajectory satisfies the three objectives.

The next step is to check if the desired state trajectory is controllable (strongly controllable). Unfortunately, there are two internal transitions into states not on the desired state trajectory (1', 2'). We can not control these transitions to occur because they are spontaneous. Thus it is impossible to design a discrete event controller satisfying the desired state trajectory.

However, if water can be supplied or stopped before those states are reached, then the two internal transitions, 1' and 2' can be controlled not to occur. It means that the controls depend on time. By Def. 2, we know the desired state trajectory is weakly controllable. Usually, controllability means strongly controllable unless explicitly specified.

In the following section, we assume that the time interval necessary to stop the water flow is sufficiently small, compared to the time interval that the water tank overflows from the position of the high sensor. Of cause, we assume that the opposite case is also true. Then, we know that the internal transitions 1' and 2' could not occur physically and be eliminated from the plant DEVS. Thus, the desired state trajectory is controllable with the assumption.

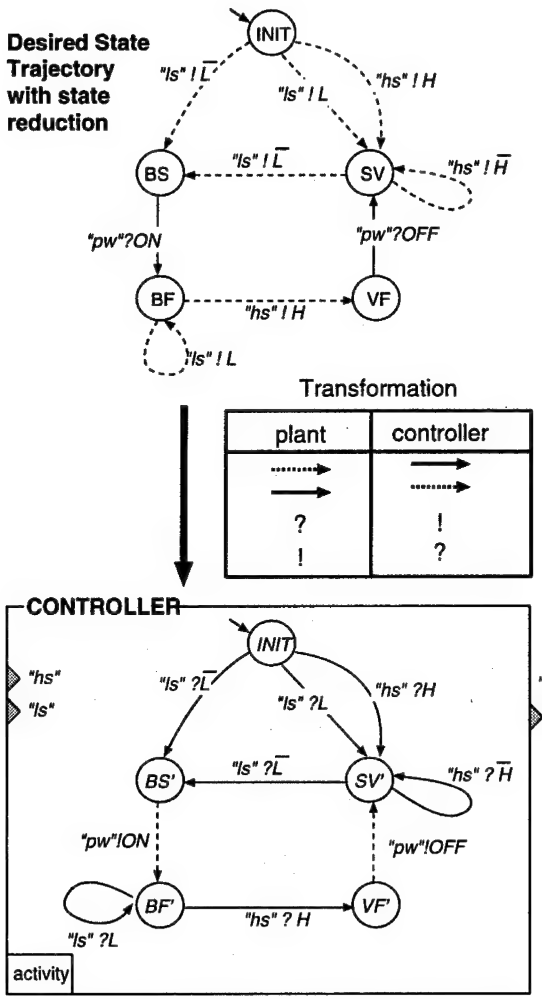


Figure 7: Design of a DES controller

7.4 Getting a discrete event controller

Until now, we specified a discrete event plant in DEVS, obtained GSTD, re-described the control objectives based on the plant DEVS, and got a desired state trajectory satisfying the control objectives. Now, we can get a discrete event controller from the trajectory using inverse DEVS transformation.

Before the transformation, we can reduce the states of the desired state trajectory. The upper part of Fig. 7 is obtained by suitable state reduction from the trajectory in Fig. 6.

A discrete event controller is obtained by inverse DEVS transformation to the desired state trajectory with reduced states shown in the upper part of the Fig. 7. The lower part of Fig. 7 shows the discrete event controller after the transformation. Note that the transformation is so straightforward and intuitive.

The three subsystems, the controller, the water pump, and the water tank are to be connected through ports that have the same name, which results in an overall system. If we analyze the dynamics of the overall system, we know that it follows the desired state trajectory only with concurrent events.

8 Conclusion

The paper proposes a new approach for analysis of discrete event systems and design of discrete event system controllers based on the DEVS formalism. Controllability and concept of inverse DEVS are defined and introduced. These two concepts play key roles in designing a discrete event controller.

A DES controller designed under the framework will be simulated in a straightforward way using DEVSim++[4] to analyze the performance of the system. But it was omitted here.

More formal description of the methodology and extension to timed systems are remained as further research.

References

[1] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes", *SIAM J. Control and Optimization*, vol. 25, no. 1, pp. 206–230, Jan. 1987.

[2] B. P. Zeigler, *Multifacetted Modeling and Discrete Event Simulation*, Academic Press, Orlando, FL, 1984.

[3] Cristos G. Cassandras, *Discrete Event Systems*, IRWIN, 1993.

[4] Tag G. Kim and Sung B. Park, "The DEVS formalism: Hierarchical modular systems specification in C++", *Proc. of the 1992 European Simulation Multiconference*, pp. 152–156, 1992.

[5] Bernard P. Zeigler, *Theory of Modelling and Simulation*, John Wiley & Sons, Inc., 1985.

Performance Modeling and Analysis of Distributed Access Network System Using DEVSim++ *

Kyou Ho Lee

Dept. of Broadband Comm. Network
ETRI
POB 106, Yusong-Gu
Taejon 305-600, Korea
kyou@winky.etri.re.kr

Tag Gon Kim

Dept. of Electrical Engineering
KAIST
373-1 Kusong-Dong, Yosong-Gu
Taejon 305-701, Korea
tkim@ee.kaist.ac.kr

Abstract

DEVSim++ is a C++ based, object-oriented modeling/simulation environment which realizes the hierarchical, modular DEVS formalism for discrete event systems specification. This paper describes a methodology for performance modeling and analysis of a distributed access network system under development within the DEVSim++ environment. The methodology develops performance models for the system using the DEVS framework and implement the models in C++. Performance indices measured are the length of queues located at connection points of the system and cell waiting times with respect to QoS grades for a network bandwidth of 155 Mbps.

1 Introduction

ATM technology based B-ISDN has been expected as a next generation high speed communication. The technology will provide end users with a variety of public services which satisfy different service requirements, traffic characteristics, and geographical coverage. An interface technique between end users and ATM local exchanges is one of major issues for the ATM network. The reference model defined by ITU-T SG13 consists of three area networks of B-ISDN UNI, namely, Customer Premises Network, Access Network, and Transport Network[2]. We have proposed a distributed access network architecture as an introductory phase of B-ISDN[5], which covers urban areas having various traffic characteristics and service requirements. The proposed system now is under development.

Performance modeling and simulation analysis are essential to optimizing system parameters for new design as well as existing ones. Especially, as complexity of systems is increased, simulation modeling may be the only means to evaluate performance of such systems. ATM networks are an example of such complex systems[1][3].

Discrete event simulation has been widely used as a performance evaluation means in many areas of system design including communication networks. In such performance study, simulation models are much more reliable and accurate than analytical ones, which may omit some aspects of the behavior of systems under design. In particular, when temporal issues of systems are significant, discrete event modeling and simulation can be considered the best solution.

The DEVS formalism developed by Zeigler supports specification of discrete event systems in hierarchical, modular manner[6]. DEVSim++ is a realization of the DEVS formalism in C++, which provides modelers with facilities for modeling systems within DEVS semantics and simulating DEVS models in hierarchical fashion[7].

This paper describes performance modeling and simulation analysis for the distributed access network system under development. The modeling methodology is based on Zeigler's DEVS formalism to exploit compatibility between the hierarchical, modular model specification and the hierarchical distributed

0-8186-6440-1/94 $04.00 © 1994 IEEE

access network system architecture.

We organize this paper as follows. Section 2 presents a brief review of the DEVS formalism and DEVSim++ modeling and simulation environment. Section 3 describes characteristics of the distributed access network system architecture. Development of a simulation model for the distributed access network system is given in Section 4 and simulation results in Section 5. We conclude this paper in Section 6.

2 DEVS Formalism: A brief review

A set-theoretic formalism, the DEVS formalism, specifies discrete event models in a hierarchical, modular form. Within the formalism, one must specify 1) the basic models from which larger ones are built, and 2) how these models are connected together in hierarchical fashion. A basic model, called an atomic model (or atomic DEVS), has specification for dynamics of the model. An atomic model AM is specified by a 7-tuple [Zeg84]:

$$AM =< X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$$

X : input events set;
S : sequential states set;
Y : output events set;
$\delta_{int}\ :\ S \rightarrow S$: internal transition function;
$\delta_{ext}\ :\ Q \times X \rightarrow S$: external transition function;
$\lambda : S \rightarrow Y$: output function;
$ta : S \rightarrow Real$: time advance function,

where $Q = \{(s,e) \mid s \in S, 0 \leq e \leq ta(s)\}$:
total state of M.

The second form of the model, called a coupled model (or coupled DEVS), tells how to couple (connect) several component models together to form a new model. This latter model can itself be employed as a component in a larger coupled model thus giving rise to construction of complex models in hierarchical fashion. A coupled model CM is defined as [Zeg84]:

$$CM =< X, Y, M, EIC, EOC, IC, SELECT >$$

X : input events set;
Y : output events set;
M : DEVS components set;
$EIC \subseteq CM.IN \times M.IN$:
external input coupling relation;
$EOC \subseteq M.OUT \times CM.OUT$:
external output coupling relation;
$IC \subseteq M.OUT \times M.IN$:
internal coupling relation;
$SELECT : 2^M - \emptyset \rightarrow M$: tie-breaking selector,

where the extensions .IN and .OUT represent the input ports set and output ports set of respective DEVS models.

DEVSim++ is a realization of the DEVS formalism in C++. The DEVSim++ environment supports modelers to develop discrete event models using the hierarchical composition methodology in an object-oriented framework. The environment is a result of the combination of two powerful frameworks for systems development: the DEVS formalism and the object-oriented paradigm.

3 Characteristics of Distributed Access Network System

A distributed access network system is an interface system between the local exchange and subscribers. The system consists of a head node, a collection of rings, each consisting of a collection of ring nodes. Each ring node is connected to a number of subscribers.

The head node mainly performs a traffic switching among ring nodes and local exchanges. Each link of the head node is based on a STM-1 frame with 155 Mbps bandwidth. The ring node mainly functions multiplexing the traffic from subscribers to the head node as well as distributing the traffic from the head node to subscribers through the ring. The traffic from subscribers is based on ATM cells with the speed of DS-1, DS-3 or STM-1 depending on applications. For the transmission speed, we consider to transform the bandwidth into a number of cells. The STM-1 frame is recommended to have 260 x 9 Bytes without overheads. One ATM cell has 53 Bytes without overheads. Therefore 44 ATM cells are included in a STM-1 frame. The transmission mechanism in the distributed access network system is shown in Figure 1. A cell-by-cell mechanism is used for adding from and dropping into subscribers in the ring node and switching in the head node. But the transmission on the ring and the network is based on the STM-1 frame.

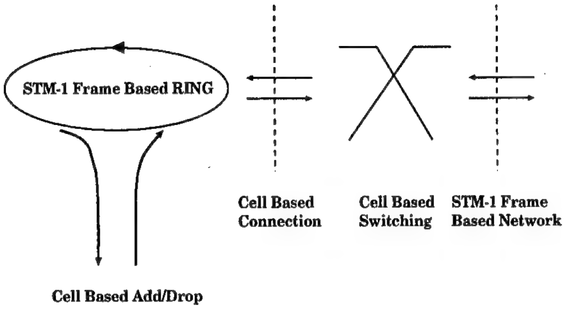


Figure 1: Transmission Mechanism

4 Models Development

This section describes modelling system architecture and shows development of a distributed access network system simulation model in DEVSim++.

4.1 Modeling Overview

The overall distributed access network system architecture is shown in Figure 2. At the top level, the distributed access network system consists of two subsystems, a HEAD and a RING. Having the (n x n) switching function for traffic, the HEAD can connect n RING's and communicate with n Local Exchange sites. Each RING comprise a set of identical Ring Nodes (RN's), each of which has 4 inputs and 4 outputs for communicating with subscribers.

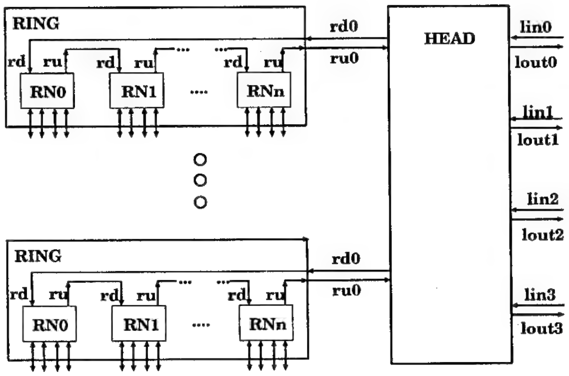


Figure 2: The System Architecture

A RN, as shown in Figure 3, consists of a ring access (RA) which accesses the ring to add or drop cells, UPWARD for concentrating cells sent from 4 subscribers into RA and DOWNWARD for distributing cells dropped from RA into 4 subscribers.

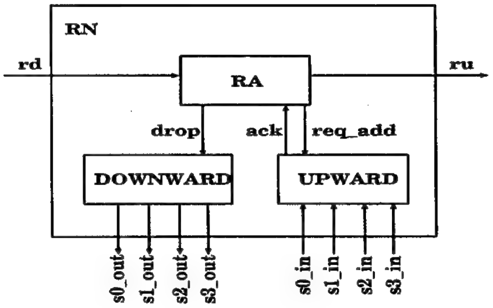


Figure 3: Coupled Model of RN

RA consists of two atomic models, D_1-to-2 and M_2-to-1, as shown in Figure 4. D_1-to-2 forwards traffic to the ring if there is no cell dropped into local subscribers. Otherwise, D_1-to-2 drops the cell to local subscribers. Likewise, M_2-to-1 forwards traffic, arrived from D_1-to-2, into the ring if there is no cell to add on. Being ready to add on the ring, M_2-to-1 inserts a cell being ready into the empty slot on the frame. If there is no empty slot on the frame, M_2-to-1 forwards with no adding.

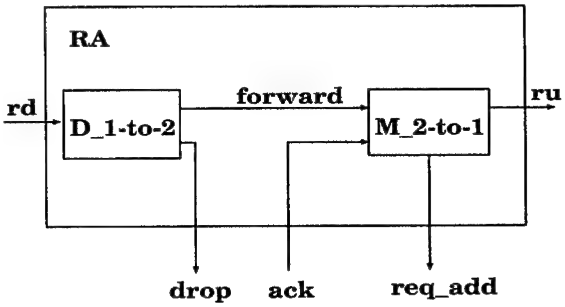


Figure 4: Coupled Model of RA

HEAD also includes RA which accesses the ring. RA in HEAD has input and output channels to receive and transmit the cell stream from and to SWITCH. On the other hand, RA in RN has I/O channels to add and drop a cell from and to subscriber.

4.2 Models Development in DEVSim++

Regarding models development in DEVSim++, we describe development of an atomic model, M_2-to-1, and a coupled model, RA, in DEVSim++.

4.2.1 Atomic Models

The atomic model M_2-to-1 can be represented in DEVS semantics as follows:

$X = \{?\text{forward}, ?\text{ack}\}$
$Y = \{!\text{ru}, !\text{req\_add}\}$
$S = \{ \text{phase} \mid \text{phase} \in$
$\quad \{\text{WAIT, ACTIVE, SEND, ADD}\}\}$

d_{ext}: $d_{ext}((\text{WAIT}), ?\text{forward}) = \text{ACTIVE}$
$\quad d_{ext}(\text{ADD}, ?\text{ack}) = \text{SEND}$

d_{int}: $d_{int}(\text{SEND}) = \text{WAIT}$
$\quad d_{int}(\text{ACTIVE}) = \text{ADD}$
ta : ta(ADD) = infinity
ta(WAIT) = infinity
ta(ACTIVE) = active_time
ta(SEND) = sending_time

O : O(ACTIVE) = !req_add
O(SEND) = !ru

Figure 5 shows the state transition diagram of M_2-to-1. M_2-to-1 has two inputs, forward and ack, and two outputs, ru and req_add. When an input arrives at the port "?forward", $M\_2 - to - 1$ transits into the phase ACTIVE and stays there for active_time units. Then it outputs on the port "!req_add" and then transits to the phase ADD. At the phase ADD, it waits for an input "?ack" to be arrived. On receiving the input "?ack", M_2-to-1 transits to the phase SEND and stays there for sending_time units. After then it returns to the phase WAIT after generating an output on the port "!ru".

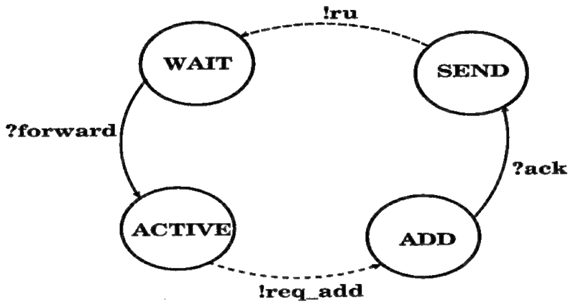


Figure 5: State Diagram for M_2-to-1

The followings are codes for implementation of $M\_2 - to - 1$ within DEVSim++.

```
const int M21_ATV_TIME = 0;
const int M21_SND_TIME = 0;
enum {WAIT, ACTIVE, ADD, SEND};

// external transition function
void m21_ext_transfn(State_vars& s,
      const timeType&, const Messages& x)
{
   if (*x.get_port() == "forward") {
      if (s.get_value("phase") == WAIT) {
         s.set_value("phase", ACTIVE);
         s.set_value("size", x.get_value());
      } else
         exit(1);
   } else if (*x.get_port() == "ack") {
      if (s.get_value("phase") == ADD) {
               int global, local;
         global = s.get_value("size");
         local = x.get_value();
         s.set_value("phase", SEND);
         s.set_value("size", global + local);
      } else
         exit(1);
   } else
      exit(1);
}

// internal transition function
void m21_int_transfn(State_vars& s)
{
   if (s.get_value("phase") == ACTIVE)
      s.set_value("phase", ADD);
   else if (s.get_value("phase") == SEND)
      s.set_value("phase", WAIT);
   else
      exit(4);
}

// output function
void m21_outputfn(const State_vars& s,
         Messages& message)
{
   int total;
   if (s.get_value("phase") == ACTIVE) {
      total = s.get_value("size");
      message.set("req_add",
               MAXCELLS - total);
   } else if (s.get_value("phase") == SEND) {
      total = s.get_value("size");
      message.set("rout", total);
   }
}

// time advance function
timeType m21_time_advancefn(const State_vars& s)
{
   if(s.get_value("phase") == ACTIVE)
      return M21_ATV_TIME;
   if(s.get_value("phase") == SEND)
      return M21_SND_TIME;
   else
      return infinity;
}
```

```
// routine for creating the model
void create_m21(Atomic_models& m21)
{
   String* name = m21.get_name();

   m21.set_sigma(infinity);

   m21.set_state_var(3,"phase","name","size");
   m21.set_state_value("phase", WAIT);
   m21.set_state_value("name", name);
   m21.set_state_value("size", 0);

   m21.set_ext_transfn(m21_ext_transfn);
   m21.set_int_transfn(m21_int_transfn);
   m21.set_outputfn(m21_outputfn);
   m21.set_time_advancefn(m21_time_advancefn);
}
```

4.2.2 Coupled Models

The coupled model RA, shown in Figure 4, consists of three atomic models. The coupled model RA can be represented in DEVS semantics as follows:

$$DN = < X, Y, M, EIC, EOC, IC, SELECT >$$
$$X = \{?nq,\ ?rd\}$$
$$Y = \{!ru,\ !drop\}$$
$$M = \{D\_1 - to - 2,\ M\_2 - to - 1\}$$

$$EIC = \{(RA.rd,\ D\_1 - to - 2.rd),\\ (RA.ack,\ M\_2 - to - 1.ack)\}$$

$$EOC = \{(D\_1 - to - 2.drop,\ RA.drop),\\ (M\_2 - to - 1.req\_add,\ RA.req\_add),\\ (M\_2 - to - 1.ru, RA.ru\}$$

$$IC = \{(D\_1 - to - 2.forward, M\_2 - to - 1.forward)\}$$

The following codes show DEVSim++ implementations for the coupled model RA.

```
void create_D12(Atomic_models& D12);
void create_M21(Atomic_models& M21);
void create_GEN(Atomic_models& GEN);

void make_RA(Coupled_models& ra)
{
   Atomic_models& d12 =
        *(new Atomic_models("D12"));
   Atomic_models& m21 =
        *(new Atomic_models("M21"));
   create_D12(d12);
   create_M21(m21);

   ra.add_inports(2, "rd", "nq");
   ra.add_outports(2, "ru", "drop");
   ra.add_children(2, &d12, &m21);
   ra.add_coupling(&ra, "rd", &d12, "rd");
   ra.add_coupling(&ra, "ack", &m21, "ack");
   ra.add_coupling(&m21, "ru", &ra, "ru");
   ra.add_coupling(&d12, "forward", &m21,
        "forward");
   ra.add_coupling(&d12, "drop", &ra, "drop");
   ra.add_coupling(&m21,"req_add", &ra,
        "req_add");
}
```

5 Simulation Experiments and Results

5.1 Simulation Experiments

Two goals for simulation experiments are as follows:

1. to foresee the maximum lengths of queues at: INBUF, CELLPOOL, RA and SWITCH. These give us important data for cell waiting status during transmission.

2. to estimate average waiting times of cells with respect to QoS grade levels, which are waiting in CELLPOOL.

For the experimentations, several cases of subscribers having different average bandwidths are applied. Since the transmission speed through a RING or a SWITCH is upto 155Mbps, if 4 RNs are connected to one RING and 4 subscribers are included in a RN, about 10 Mbps in average can be given to one subscriber. Maximum queue lengths and average waiting time are measured for various subscribers' bandwidths ranging from 5Mbps to 100Mbps.

A summary of assumptions for simulation modeling is as follows:

1. 90% of the traffic from a RING is routing to the network through the SWITCH. And the rest(10%) is forwarding back into the same RING, which is destined to the subscribers connecting to the same RING.

2. The traffic given at any port of the SWITCH are divided and routed to the rest ports of the SWITCH with equal probability.

3. Any RN has statistically the same portion of traffic sent from or added into a RING. If a RING includes 4 RNs, 25% of the traffic sent from a RING are dropped to be routed into destined subscriber. The rest are forwarded into the next stage of a RN. During forwarding, a new traffic from subscribers is added on, which has the same probability as dropping.

For simulating cell loss rate of 10^{-12}, more than 10^{12} cells should be generated. A couple of techniques, such as importance sampling[8] or the generalized extreme value theory[9], has been proposed to deal with such a problem.

One way is that the value for numbers of cells, instead of cell by cell, are generated and distributed with given probability density functions. It is an easier way to handle event messages as well as to implement simulator. Instead of counting how many events("cells") waiting in queues, we just consider the integer value calculated in queues.

We employed a token passing based simulation scheme. In the scheme, only one token traverses each RING. Each token consists of a number of slots. Indeed, a slot means a message. When a model receives a token, it can remove/insert messages from/into the token. But, the total number of slots in a token cannot exceed a bound. We have already known that 44 slots exist in a frame($125\mu s$) of an 155Mbps RING. It is natural that a token is responded by n*44 slots. For simplicity, we set n to 7. Consequently, a token is composed of 44*7 slots.

The relationship between physical and virtual times can be acquired easily. Let the RING turnaround time in virtual time be T_r. Thus, one unit in virtual time corresponds to $125*7/T_r$ μs. Now, we should discuss about how we can design subscriber models with given average and maximum bandwidths. Consider that in a RING only one subscriber is active and others are inactive. Since 155Mbps corresponds to 44*7 cells during T_r, bandwidth of ω corresponds to $44*7/155*\omega$ cells. For reducing simulation time complexity, we assume that a subscriber generates cells in a burst manner. Therefore, if a subscriber generates α cells at an instance, bandwidth of ω corresponds to $44*7/155*\omega/\alpha$ times of burst output generation frequency during T_r. Then, the intergeneration time ta is defined as:

$$ta = \frac{N}{\frac{7*44}{155} * \frac{\omega}{\alpha}} = \frac{155}{7*44} * \frac{N*\alpha}{\omega}.$$

We set that $T_r = 1$ and $\alpha = 44*7$. Consequently,

$$ta = \frac{155}{\omega}.$$

Assume that the request rate of a subscriber has a uniform distribution and the maximum and average bandwidths of the subscriber is ω_{max} and ω_{avg}, respectively. Then the subscriber can be modeled statistically as:

$$U[\frac{155}{\omega_{max}}, 2*\frac{155}{\omega_{avg}} - \frac{155}{\omega_{max}}]$$

where $U[a,b]$ denotes a uniform random number generator in $[a,b]$.

5.2 Simulation Results

Table 1 shows maximum queue lengths for the given subscriber's average bandwidths. Each number means how many cells are waiting in the queue. In other words, it gives the queue length which should be implemented to avoid cell loss.

Table 1: Maximum Queue Length

ω_{avg}(Mbps)	inbuf	cellpool	head	switch
5	308	655	30	280
10	308	3552	171	343
30	770	36334	282	347
50	7392	45584	263	345
100	27643	45815	319	340

ω_{max}(Mbps) = 155.

The simulation results for average waiting times with respect to QoS grade levels are shown in Table 2. Note that the traffic with lower QoS grades can rarely be served for.

Table 2: Average Waiting Time

ω_{avg}(Mbps)	QoS0	QoS1	QoS2	QoS3
5	1.96	1.72	1.66	1.61
10	20.79	7.05	4.56	2.97
30	∞	∞	64.60	7.47
50	∞	∞	∞	33.64
100	∞	∞	∞	68.17

ω_{max}(Mbps) = 155.

6 Conclusion

Performance modeling and analysis for the distributed access network system under development has been discussed. The objectives of modeling are not only to analyze dynamic traffics in a transient state but also to make decisions of architectural parameters such as queue lengths. By consideration of the distributed access network system architectural characteristics, we employ Zeigler's DEVS formalism and develop model within DEVSim++ environment. As results of simulation experiments in DEVSim++, we analysis the length of queues located in connection points. Also we analysis cell waiting times with respect to QoS grade levels, which are for the cells waiting for to be added on a network.

Such results help us to decide the maximum lengths of queues to avoid cell loss. We can observe that a queue in the SWITCH is rarely dependent of the subscriber's bandwidth. But queues at the other locations in the RING is much dependent of each subscriber's bandwidth.

We also observe that the traffic with lower QoS grade can rarely be served if a subscriber's bandwidth is more than 30 Mbps.

For future work, we should collect more data for various situations. From this we can optimize the design parameters for the system under development.

Acknowledgement

We give special thanks to Dr. Mun-Kee Choi and Mr. Tae-Soo Chung for their guidence. We are also very grateful of our colleagues for their helpful comments.

References

[1] A. Chai and S. Ghosh, "Modeling and Distributed Simulation of a Broadband-ISDN Network," *IEEE Computer*, September 1993.

[2] W. Stalling, *ISDN and Broadband ISDN*, Macmillan Publishing Company, 1992.

[3] G. Pujolle and D. Gaiti, "Performance Management Issues in ATM Networks," *Proceedings of Information Networks and Data Communications*, April 1994.

[4] V. S. Frost and B. Melamed, "Traffic Modeling For Telecommunications Networks," *IEEE Communications Magazine*, March 1994.

[5] E. Son, S. Hong, and K. Kim, "Network Architecture for the Introductory Phase Broadband Subscriber Access Network," *Proceedings of Information Networks and Data Communications*, April 1994.

[6] B. P. Zeigler, *Multifacetted Modeling and Discrete Event Simulation*, Academic Press, 1984.

[7] T. Kim and S. Park, "The DEVS Formalism: Hierarchical Modular Systems Specification in C++," *Proceedings of the 1992 European Simulation Multiconference*, June 1992.

[8] Q. Wang and V. S. Frost, "Efficient Estimation of Cell Blocking Probability for ATM Systems," *IEEE Trans. on Networking*, April 1993.

[9] F. B. Bernabel, "ATM System Buffer Design Under Very Low Cell Loss Probability Constraints," *Proceedings of IEEE Conf. on Computer Communication, INFOCOM'91*, April 1991.

Session 2E:

DEVS Workshop Working Session

Session 2F:

Applications I

SmartDb: An Object-Oriented Simulation Framework for Intelligent Vehicles and Highway Systems *

Aleks Göllü, Akash Deshpande, Praveen Hingorani, and Pravin Varaiya
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720

Abstract

SmartDB is a framework for the uniform specification, simulation, optimization, evaluation, and implementation of Intelligent Vehicle Highway System (IVHS) alternatives. The salient concepts in SmartDB are: 1) layered control architecture, 2) coordination of distributed control agents through communication, 3) combined discrete and continuous dynamical systems, known as hybrid systems, and their control and verification, 4) object oriented simulation, and 5) distributed and open architecture. This paper summarizes the first three concepts and describes in detail the simulation constructs and the distributed and open architecture of SmartDB.

1 Introduction

Highway congestion is imposing an intolerable burden on many urban residents. It is estimated that lost productivity due to traffic congestion costs $100 billion each year in the United States. Alongside congestion, safety continues to be a prime concern. In 1991, 41,000 persons died in traffic accidents, and more than 5 million persons were injured.

Intelligent Vehicles and Highway Systems (IVHS) is a comprehensive program initiated by the U.S. Government under the Intermodal Surface Transportation Efficiency Act of 1991 to improve safety, reduce congestion, enhance mobility, minimize environmental impact, save energy, and promote economic productivity in the transportation system. The IVHS program combines several modern technologies, including information processing, communications, control, and electronics. IVHS has the following sub-programs.

Advanced Traffic Management Systems
ATMS provides subsystem integration of traffic management and control systems, and performs real-time traffic control to respond to dynamic traffic conditions.

Advanced Traveler Information Systems
ATIS acquires and analyzes information about transportation network dynamically, and communicates advisory information to travelers.

Advanced Vehicle Control Systems
AVCS uses computers, communications, and control systems in the vehicles and the highways to enhance vehicle control.

Commercial Vehicle Operations
CVO improves the safety and efficiency of commercial vehicle and fleet operations.

Advanced Public Transportation Systems
APTS integrate public transportation with vehicle-highway systems by using component technologies from other functional areas.

Apart from the U.S., there is substantial IVHS activity in Europe under the PROMETHEUS[1] and the DRIVE[2] projects, and in Japan under the RACS[3], AMTICS[4] and VICS[5] projects.

*Research supported in part by the National Science Foundation and the California PATH program in cooperation with the State of California and US DOT. The contents of this report reflects the views of the authors who are responsible for the facts and accuracy of the data presented herein

[1] Program for European Traffic with Highest Efficiency and Unprecedented Safety

[2] Dedicated Road Infrastructure for Vehicle Safety in Europe

[3] Road/Automobile Communication System

[4] Advanced Mobile Traffic Information and Communication System

[5] Vehicle Information and Communication System

0-8186-6440-1/94 $04.00 © 1994 IEEE

Modeling and simulation have been identified as important steps in realizing these transportation initiatives. The IVHS strategic plan [4] requires modeling and simulation in the following areas: urban traffic network models, traffic system models, vehicle-road models, driver-vehicle models, traffic models with dynamic traffic assignment, driving scenario simulation, and advanced vehicle control systems (AVCS) architecture simulation. A framework in which IVHS alternatives can be specified, simulated, and evaluated uniformly is crucial for objective comparison of the proposed alternatives. Such a framework must also aid in the implementation of the selected alternative.

SmartDB is a framework for uniform specification, simulation, optimization, evaluation, and implementation of IVHS alternatives. *SmartDB* is targeted at the Advanced Vehicle Control Systems (AVCS) functions. At the same time, it also addresses the automation requirements of the other functional areas. In particular, *SmartDB* meets the modeling and simulation needs of urban traffic network models, traffic system models, vehicle-road models, traffic models with dynamic traffic assignment, and AVCS architecture simulation.

The concepts for *SmartDB* have emerged from the SmartPath project at the California PATH Laboratory at the University of California, Berkeley [3]. *SmartDB* is an object-oriented distributed processing simulation environment that can be scaled to meet the performance requirements of large-scale applications. *SmartDB* is an open system that can interface with other simulation environments. *SmartDB* decomposes the IVHS modeling and simulation problem into the following stages:

1. parametrized modeling of the physical system and the control agents in an object oriented semantic data model,

2. simulation of the discrete and continuous behavior of all the objects in the model, and optimization of the model parameters,

3. evaluation of system performance according to specified criteria, and

4. model validation and implementation of selected control strategies for deployment.

These stages are shown in Figure 1.

1.1 Semantic Data Model

The *SmartDB* semantic data model captures the relevant aspects of the physical world in a logical model. The logical model consists of software objects that represent physical components. For example, it models vehicles, highway segments, engines, brakes, sensors, and other physical system components as objects. The objects in the logical model have semantic content corresponding to their characteristics, interrelationships, constraints, and behaviors.

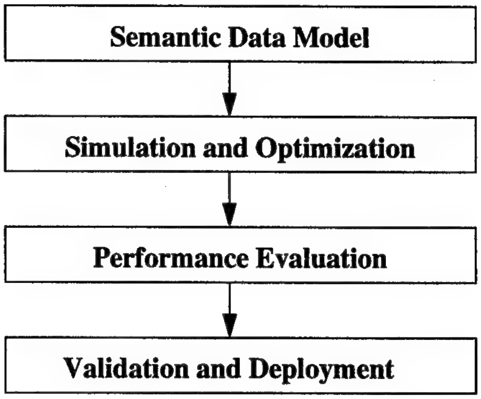


Figure 1: *SmartDB* Stages.

1.2 Simulation and Optimization

A simulation run occurs when all objects in a given highway and traffic configuration execute their state evolution behavior. Since object characteristics are parametrized, the model parameters can be tuned during a simulation run based on specified optimization criteria. Such a tuning leads to an on-line optimization of the control strategy.

1.3 Evaluation

The state evolution of the objects during a simulation run can be observed using specialized monitor objects. The monitor objects generate performance results based on specified evaluation criteria. These results are used for an objective evaluation and comparison of alternative control strategies for IVHS. Once a control architecture is selected, *SmartDB* can be used for its rigorous simulation, optimization, and implementation.

1.4 Validation and Deployment

SmartDB's object-oriented approach simplifies model validation and modular deployment. Models for physical objects such as sensors and communication equipment can be validated by replacing software objects in the logical model by the corresponding physical components. *SmartDB* behavior in the two cases—using the software objects and using the physical components—can then be compared to validate the logical model and the controller design.

Similarly, *SmartDB*'s object-oriented approach simplifies modular deployment of the selected control architecture. Once the selected control architecture is simulated and optimized, each software control object that models the behavior of a specific control agent in the physical world can be implemented independently on physical components for deployment.

2 Concepts and Functions

2.1 *SmartDB* Concepts

The salient concepts in *SmartDB* are given below:

1. layered control architecture,
2. coordination of distributed control agents,
3. combined discrete and continuous dynamical systems, known as hybrid [8] systems, and their control and verification,
4. object-oriented simulation, and
5. distributed and open architecture.

2.1.1 Layered Control Architecture

In the layered control architecture proposed by Varaiya and Shladover [7, 6], vehicles perform simple maneuvers such as merging into platoons, splitting from platoons, following the leader, changing lanes, and entry and exit. A vehicle accomplishes complex end-to-end trajectories by performing a sequence of such simple maneuvers. Efficient transportation throughput is achieved by tuning traffic parameters such as platoon size and vehicle speed. The control strategies for such behavior are organized into the following layers: regulation layer, coordination layer, link layer, and network layer. These layers are shown in Figure 2.

Given a maneuver to perform, the vehicle follows a control strategy that regulates its dynamical behavior to a trajectory permitted by that maneuver. Such control strategies constitute the regulation layer. The maneuver to be followed by a vehicle at a given time is determined by coordinating with other vehicles in the neighborhood. The control strategies used for such coordination constitute the coordination layer. The control strategies adapt their behavior based on information about highway traffic conditions. The traffic conditions on highway segments are monitored and controlled by road-side control elements. These are collectively known as the link layer. Finally, information from individual highway segments is aggregated, and end-to-end routing and congestion control is accomplished in the network layer.

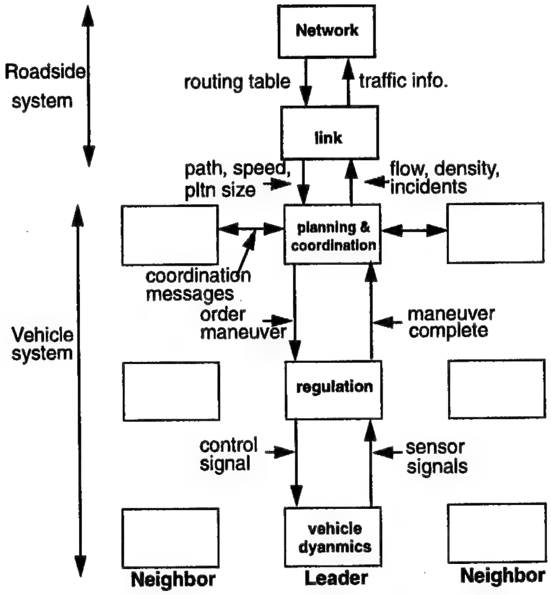


Figure 2: Layered Control Architecture.

SmartDB allows the specification of this and other layered control architectures.

2.1.2 Coordination of Distributed Control Agents

The layered scheme described above yields a distributed control strategy since each vehicle and each highway segment is responsible for its own control. At the same time, effective coordination of these distributed control agents is essential for efficiency and safety. The agents coordinate by following simple heuristic rules. For example, when a vehicle senses another vehicle ahead of it, it requests a merge with it to form a platoon. Forming a platoon increases the efficiency of the highway. However, if the leading vehicle is already a part of a large platoon, it may refuse the merge request since inordinately large platoons are potentially unsafe. Such coordination strategies are modeled using a Discrete Event System (DES) approach. The control agents communicate the discrete events to each other based on coordination protocols. Thus, communication mechanisms are essential both for gathering sensory information and for executing these coordination protocols.

SmartDB allows the specification of sensors, transmitters and receivers, and of communication protocols.

2.1.3 Verification and Control of Hybrid Systems

Whereas the coordination strategies deal with discrete events, regulation strategies deal with continuous evolution. For example, if a merge maneuver is to be executed, then the regulation layer controller must first accelerate the vehicle, close the distance between itself and the vehicle ahead of it, and finally decelerate and follow at the same speed while maintaining a safe distance in between. It is clear that acceleration and braking, speed and distance are continuous parameters that evolve in continuous time. Thus, the discrete coordination event corresponding to the merge command, and the continuous regulation law corresponding to the merge trajectory must be dealt with together. A hybrid system approach is used to model the combined discrete and continuous behavior.

SmartDB allows the specification of both discrete and continuous behavior.

2.1.4 Object-Oriented Simulation

SmartDB is an object-oriented [1, 2] software framework [5]. The object oriented approach is used to construct a logical model of the physical components and their control agents. The objects in the logical model have semantic content corresponding to their characteristics, inter-relationships, constraints, and behaviors. The object-oriented approach simplifies model validation and system implementation for deployment. The object-oriented model is described in section 3.1.

2.1.5 Distributed and Open Architecture

SmartDB is a distributed processing simulation environment that can be scaled to meet the performance requirements of large-scale applications. *SmartDB* is an open system that can interface with other simulation environments. The system architecture is described in section 3.3.

2.2 *SmartDB* Functional Categories

SmartDB is designed to perform the following functions:

Configuration Management—
the ability to specify a highway network configuration, the traffic patterns on it, and the vehicle and traffic control strategies;

Fault Management—
the ability to detect faults and significant events such as accidents and congestion, and to respond to them with graceful degradation of highway performance and with automatic fault recovery;

Performance Management—
the ability to track, optimize, and fine-tune the transportation system performance;

Planning Management—
the ability to specify and simulate alternative highway and traffic configurations and control strategies for the purpose of planning;

Resource Management—
the ability to provide an inventory of all highway and vehicle resources and to schedule them for preventive maintenance;

Accounting Management—
the ability to specify tolls and taxes, and to account for highway usage;

System Management—
the ability to manage the resources of the *SmartDB* system for multi-user and multi-processor operation.

3 *SmartDB* Implementation

3.1 Object Architecture

All *SmartDB* objects consist of state, state evolution, interface, input, and output components. This is shown in Figure 3. An object's external interface is defined by its input and output specifications. So long as this interface is met, the state and state evolution of an object can be reimplemented.

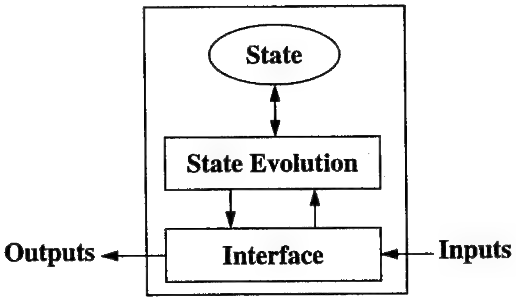


Figure 3: *SmartDB* Object

SmartDB objects are arranged according to a set of connection and containment rules to create aggregate objects. The aggregate objects are also *SmartDB* objects except that their state and state evolution components are implemented by other objects.

It is envisioned that highway automation will be achieved by adding sensors, transmitters, receivers, and control modules to the highway and the vehicles.

To this end *SmartDB* provides a particular aggregate object called "Smart Object" described in Figure 4.

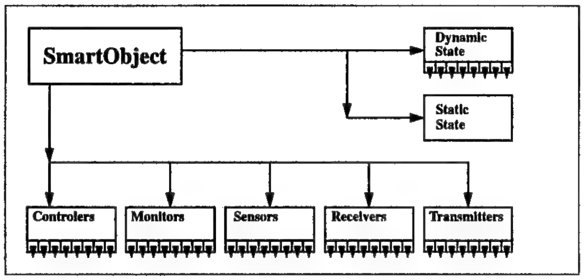


Figure 4: Uniform Representation of IVHS Objects.

Smart objects enable the user to create control and communication hierarchies in a structured manner and consist of the following components:

- static state such as lane width, and vehicle weight, that does not change during a simulation run
- dynamic state such as vehicle speed and lane density,
- state evolution behavior,
- control objects such as regulation objects that determine speed,
- monitor objects for observing state evolution, e.g., a gas tank agent that monitors the amount of carbon-monoxide produced,
- sensors for providing information about the environment, e.g., distance to vehicle in front,
- transmitters and receivers for communicating with neighboring objects,
- inputs and outputs.

3.2 *SmartDB* Functions

The current version of the *SmartDB* implementation supports the configuration, fault, and performance management functions of IVHS. We now describe how the object data model implements these categories.

3.2.1 Configuration Management

SmartDB provides a set of highway objects that can be interrelated according to a set of connection and containment rules. Any possible highway configuration can be created using these objects and their interrelationships. The transportation system is divided into zones. Each zone contains multiple highway segments interconnected using junctions. The highway segments are terminated using traffic sources and sinks. The highway segments consist of sections, entries, and exits. Junctions and sections are divided into lanes. These building blocks and the graphical editor used to create highway configurations are shown in Figure 5.

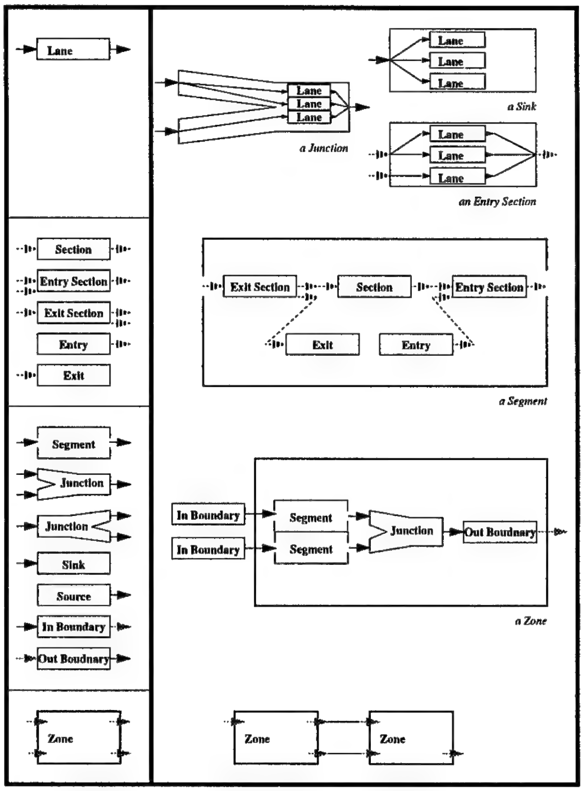


Figure 5: *SmartDB* Graphical Editor for Highway Building Blocks

Incoming traffic to the automated highway is generated by traffic generators in the source and entry objects. Traffic leaving the automated highway is absorbed by traffic absorbers in the exits and sinks. Traffic generator and absorber objects are parametrized to create different traffic patterns. When generators and absorbers are replaced by gateway objects *SmartDB* can interface with packages such as NetSim and Integration.

Vehicle and traffic control strategies are specified by configuring the control, communication, and sensor components of the relevant smart objects.

3.2.2 Fault Management

The simulation framework provides fault detection as well as fault creation mechanisms.

If an object fails to remain within system constraints corresponding fault events are created. System constraints correspond to states the system should not enter. For example a vehicle has to stay within highway boundaries, similarly two vehicles can not occupy the same space at the same time. *SmartDB* creates "accident" events when these constraints are violated. Other constraints such as maximum acceleration or lane change direction in a section can also be specified by the user. Monitor objects are used to detect the violation of constraints. For example a lane change monitor can be used to raise a fault if a vehicle performs an illegal lane change.

Faults such as communication failures or accidents can be created to ensure that the control objects can respond to them with graceful performance degradation and automatic fault recovery.

3.2.3 Performance Management

Monitor objects are used to collect statistical information about the system and to create performance reports. Monitor objects are like any other *SmartDB* objects; they observe the system evolution through their inputs, process this data through their state evolution, and output the desired statistics.

3.3 Process Architecture

In SmartDB, object state evolution is driven by passage of time or by occurrence of events. Events are generated by SmartDB objects as output messages and are communicated to the addressed objects as input messages. In this section we describe how the *SmartDB* architecture guarantees the timely evolution of each object and the timely communication of events to objects.

Time passage in *SmartDB* is represented by a global clock, which defines the smallest time step of the system. All evolution takes place at discrete advancements of this clock.

Each time driven object specifies the time step for its state evolution as a multiple of the global clock step. Rapidly evolving objects such as engines change their state more frequently, while more passive objects such as a roadside link controllers change their state at larger time steps.

Event driven objects exercise their behavior only when events are delivered to them. Coordination objects in vehicles, for example, respond to "Merge Request"s coming from other vehicles; the network layer changes the routing table upon an "accident".

Process layers are used to execute the simulation of collections of objects that evolve at same time steps. The process layers themselves can be time or event driven. Process layers are controlled by a "Process Coordinator". The process coordinator schedules the execution of time driven layers based on their time step and schedules the execution of event driven process layers if any events are raised against them.

A process layer can do the following:

- simulate the time driven evolution of all instances of an object type,
- for all instances of an object type with an outstanding event, deliver the event and simulate the event driven evolution.

If event driven objects are put in a time driven process layer, event delivery for these objects takes place only when the corresponding process layer is executed.

In a simulation run the process coordinator executes the process layers according to their time step, which in turn execute the simulation of the objects they contain.

The process architecture that implements the layered architecture proposed by Varaiya [7] is shown in Figure 6.

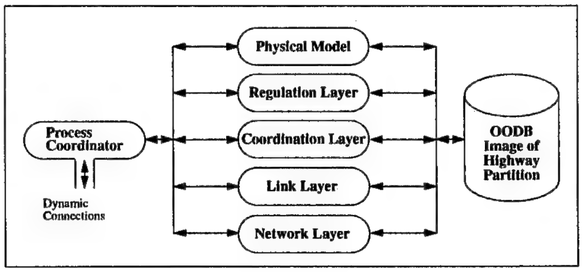


Figure 6: *SmartDB* Process Architecture.

The physical layer is time driven and simulates the time driven engine objects that generate the vehicle displacements.

The regulation layer is time driven. It contains event driven regulation supervisors and time driven maneuver objects. The supervisors switch between maneuvers based on incoming messages from the coordination layer; the maneuvers control the behavior of the gas pedal and the steering wheel.

The coordination layer is time driven. It contains event driven coordination objects. Coordination objects in different vehicles exchange messages to determine the maneuver a vehicle should execute. These decisions are communicated to regulation supervisors through messages.

The link layer is time driven and contains time driven link objects that set traffic parameters such as target speed and average platoon size in highway sections.

The network layer is event driven. It is executed only if an accident occurs. Upon an accident it reconfigures the routing tables.

The simulation objects are placed in an object-oriented database (OODB). The database provides a natural mechanism to save and store the state of a simulation.

3.4 Open and Distributed Architecture

SmartDB has an open architecture. *SmartDB* allows the user to create the desired simulation granularity by configuring a process layer architecture. For each process layer the user specifies the object types for time and/or event driven simulation. The OODB makes the simulation state visible to any user. It has a well-defined interface, and provides a default mechanism for any other simulation package to interface with *SmartDB*.

SmartDB supports distributed simulation. Zones serve as the unit of distribution: different zones can be distributed to different processors; they have their own database and their own clock. Since a vehicle can communicate with and sense other vehicles only in its neighborhood, communication between the distributed processors is restricted to objects in adjacent highway segments. Thus, locality of reference enables efficient distributed processing. Distributed simulation is depicted in figure 7. The boundary object between the last section of the previous zone and the the first section of the next zone coordinates communication between the processors, ensures synchronization of simulation clocks on different processors, and controls object migration between databases.

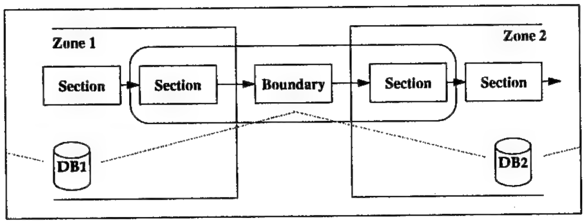


Figure 7: Distributed Simulation

4 Summary

SmartDB is a framework for uniform specification, simulation, optimization, evaluation, and implementation of IVHS alternatives.

SmartDB reduces system specification to mixing and matching of software components. These components are used to create highways, traffic patterns, and control and communication hierarchies.

A simulation run occurs when all objects in a given highway and traffic configuration execute their state evolution behavior.

The object behaviors are parametrized, and the performance of the transportation system being simulated can be optimized by adjusting these parameters based on specified optimization criteria.

The system performance can be observed and evaluated using monitor agents that collect statistical information about the system and generate performance reports. Different control alternatives are compared by simulating them with identical highway configurations and traffic patterns. The performance reports generated by the monitor agents in the respective simulations are used for objective comparison and evaluation of the alternatives.

Once a control architecture is selected, the individual software components can be implemented as physical hardware components for deployment .

References

[1] G. Booch. *Object Oriented Design with Applications.* Benjamin/Cummings, Redwood City, CA. 1991.

[2] P. Coad and E. Yourdon. *Object-Oriented Analysis.* Yourdon Press, Englewood Cliffs, NJ. 1991.

[3] F. Eskafi and P. Varaiya. Smartpath: Automatic Highway Simulator. *PATH Technical Memorandum*, UC Berkeley. June 1992.

[4] IVHS America. *Strategic Plan for Intelligent Vehicle-Highway Systems in the United States.* Report N0. IVHS-AMER-92-3. 20 May 1992.

[5] R. Jonson. How to Develop Frameworks. *OOPSLA Tutorial Notes.* ACM Press. 1993

[6] S. Shladover *et. al.* Automated Vehicle Control Developments in the PATH program. *IEEE Trans. Vehicular Tech.* Vol. 40. pp. 114-130. Feb. 1991.

[7] P. Varaiya. Smart Cars on Smart Roads: Problems of Control. *IEEE Trans. Automatic Control* Vol. 38, No 2. Feb. 1993.

[8] A. Göllü, Pravin Varaiya; Hybrid Dynamical Systems; In *Proceedings of the 28th Conference on Decision and Control,* pages 2708 2712, Tampa FL, December 1989.

Modeling the Interactive mode of SmartPath

Farokh H. Eskafi

EECS Department and PATH/ITS
University of California at Berkeley
Berkeley, CA. 94720

Delnaz Khorramabadi

EECS Department and PATH/ITS
University of California at Berkeley
Berkeley, CA. 94720 *

Abstract

SmartPath is a highway system simulator. The program can be used to test, simulate, and evaluate the performance of the designs of different modules and instrumentations like engine models, sensors, and communications. The package consists of two separate modules: simulation and animation. The simulation runs on Sun Sparc or Silicon Graphics workstations. The animation program, runs on Silicon Graphics workstations, and it produces a three-dimensional, color animation of AHS traffic. *SmartPath* could be used in two modes. In the batch mode the simulator is run first to generate data which could then be viewed using the animator. In the interactive mode, the simulator and animator run simultaneously allowing the user to control vehicle maneuvers in real time. This feature allows the system to mix 'manual' and 'automated' vehicles and to test the robustness of the control algorithms.

In this paper, we describe the modeling of basic elements of the interactive mode of *SmartPath* and the interfaces used to allow interactive control of vehicles.

1 Introduction

SmartPath is a highway system simulator. It is designed to provide a framework for simulation and evaluation of Intelligent Vehicle Highway System (IVHS) alternatives. *SmartPath* can simulate automated, manual, or mixed mode traffic; it also accommodates different control, communication, and computing architectures.

SmartPath is a micro-simulator, i.e. the functional elements and the behavior of each vehicle and highway with respect to normal and degraded mode of operations are individually modeled. Its evaluation reports are targeted to microscopic as well as macroscopic performance analysis.

At the macroscopic level *SmartPath* can be used to understand the steady-state behavior of the highway system, i.e., how the highway system would perform under various control policies in terms of highway capacity, traffic flow, and other performance measures of interest to transportation system planners and engineers.[1]

At the microscopic level it can be used to analyze the transient behavior of the highway system and to test, simulate, and evaluate the performance of different modules inside a car like sensors, vehicle engines, and communication devices. The effects of high level control policies on vehicles can be observed by tracing the trajectory of a vehicle during a simulation run.

SmartPath consists of two separate modules: simulation and animation. The *SmartPath* animator is a tool to view and examine the simulated data of the AHS in the most natural way. The simulation data provides information about the position, speed, and maneuvers of each vehicle in the AHS at every unit of simulation time. In addition, the user can select a vehicle and view the interaction between the vehicle and its neighboring vehicles. The user can control the

*Research supported in part by the California Department of Transportation, through the PATH program, and the Army Research Office under contract DAAH04-94-G-0026. The contents do not necessarily reflect the official views of the State of California.

[1] Examples of how *SmartPath* can be so used are given in [1] and [2].

0-8186-6440-1/94 $04.00 © 1994 IEEE

motion of the helicopter, rewind the animation, and adjust its speed. the motion of the helicopter can be restricted to the highway or forced to follow a specific car.

SmartPath can be used in two modes: batch, and interactive. In the batch mode, the simulator is run first to generate data which could then be viewed using the animator. In the interactive mode, the animator is synchronized with the simulation and can be used to send commands to the vehicles being simulated. This feature allows the user to control the vehicles in real time. In either mode, the user is provided with a view about neighboring traffic. This view may be varied to try to mimic different situations, for example, the view may be what is visible form the windshield. Figure 4 shows a frame of animation.

In this paper we describe the interactive mode of *SmartPath*. Section 2 discusses the multi-layer control hierarchy used to model the Automated Highway System (AHS). In this section we also briefly describe the building blocks of *SmartPath*. In section 3, we explain the interactive mode of *SmartPath*.

2 Elements of an Automated Highway System

A major objective of the AHS is to increase highway capacity and safety. This objective is achieved in part by organizing the traffic in platoons, which consist of one or more cars traveling together as a group. The first vehicle in the platoon is called a *leader*; the others are *followers*; a one-vehicle platoon is called a *free agent*. At every moment of time, a vehicle under automatic control is either a leader, a follower, or a free agent.

The control of a vehicle in AHS is carried out by the four-layer control hierarchy displayed in figure 1, which was first proposed by Varaiya and Shladover in [3].

We now discuss each layer starting from the top.

The network layer controller assigns a path to each vehicle entering the system. This assignment could be generated from a static routing technique which uses distances to find the shortest path from origin to destination, or a dynamic routing scheme using the measured and expected flow of the traffic and information about incidents to calculate the expected travel time for the vehicle. In the first case, the path can be calculated by a navigational device installed in the vehicle; in the second case, the network controller requires global information with regard to the highway system and has to be operated from a traffic management control center.

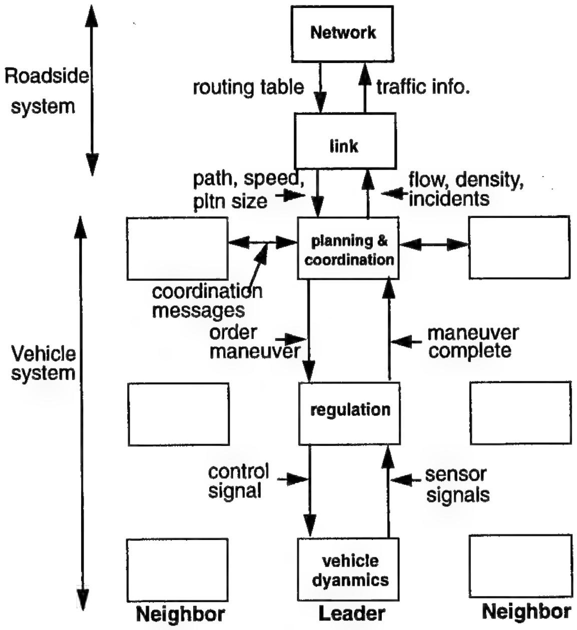


Figure 1: Control Hierarchy for Highway System

The *link layer* controllers are responsible for the smooth flow of traffic in each lane and the distribution of flow among lanes. It recommends a course of action to the vehicle, i.e., change-right, change-left, or stay-in-own-lane. The link layer controllers are on the road side, but their scope of operation is limited to a section of highway. In the present traffic condition, it is the human driver who decides which lane is more suitable. In either case, the link layer recommendations are explicit functions of capacity of the highway, flow of traffic, and destination of the vehicle.

The *coordination and planning layer* controller of a vehicle determines which of the three maneuvers – merge, split, and change lane– to attempt at any time. In the *merge* maneuver, two platoons join to form one platoon; in *split*, one platoon separates at a designated position to form two platoons, and in *change lane*, a free agent changes lane.

In order to execute a maneuver safely, the coordination layer controller initiates a structured exchange of messages—a communication *protocol*—with the neighboring vehicles. The message passing can be implicit (looking for a gap to move-in) or explicit (requesting a change-lane maneuver by transmitting a packet with the radio wave). At the end of the exchange, the coordination layer secures agreement from the neighboring vehicles for safe execution of the maneuver and instructs the regulation layer to execute the maneuver.

The *regulation layer* controller implements the requested maneuver. The control action is typically decomposed into longitudinal control which determines acceleration and braking, and lateral control which determines the steering action needed to maintain the vehicle in its lane or move the vehicle to an adjacent lane.

The *vehicle dynamics layer* in figure 1 receives steering, throttle and brake actuator commands from the regulation layer and returns information such as vehicle's speed, acceleration, engine state, etc., which are needed to implement the control actions mentioned above.

To summarize the highway control hierarchy: the network layer assigns a route, the link layer provides *en route* guidance, the coordination layer selects which maneuver to execute in order to follow the path assigned by the link layer, and the regulation layer implements the maneuver selected by the coordination layer.

SmartPath implements the control hierarchy described above by modeling two basic elements: vehicle and highway.

Vehicle A vehicle is composed of five independent and communicating modules: sensors, communications, regulation, maneuvers, and supervisor. The sensors module provides information about a vehicle's surrounding environment; the communications module provides the vehicle with facilities for transmitting to, and receiving from, neighboring vehicles and roadside link layer controllers; the regulation module implements the feedback laws; and the maneuvers and supervisor modules together implement the coordination layer.

Highway In SmartPath, a highway is defined by its length, maximum number of lanes, number of automated, manual, and transition lanes (if any), number of exits and entrances and the locations of exits and entrances. The physical topology of the highway (width of the lane, curvature of the road, etc.) is part of the specification of a highway and must be specified thoroughly.

A highway is divided into smaller structures called sections. Each section consists of a certain number of lanes. A lane has a length, width, curvature, type (which can be automated, transition, manual, entrance, or exit), and some flags which correspond to the special features that might exist in the lane, e.g. lane is blocked, or it doesn't have a right or left adjacent lane.

For a multi-highway simulation, a *junction* structure is used to define the interconnections among the lanes of connecting highways.

The internal architecture of *SmartPath*, its time and event driven simulation modes, and how evolution of individual processes is synchronized, is described in [4].

3 *SmartPath* Interactive Mode

Every vehicle in *SmartPath* operates in the "automated", "intelligent manual", or "manual" mode. By using the *SmartPath* animation interface, one can select a vehicle and change its mode of operation from one to another. The default mode is automated.

Automated (*Au*) In this mode, the vehicle has the ability to sense its neighboring vehicles, communicate with the roadside to receive the routing information, coordinate with other vehicles within its sensor range to perform maneuvers, and become a member of a platoon (as leader or follower). For a complete description of an automated vehicle, see [4, 5].

Intelligent manual (*IM*) In this mode, the vehicle operates like an automated vehicle, i.e., it communicates with other vehicles to coordinate the maneuvers, and it can become part of a platoon. However, it doesn't communicate with the linklayer controller, and the functionality of the link layer is transferred to the user, who can issue recommendations using the interface panel shown in figure 2.

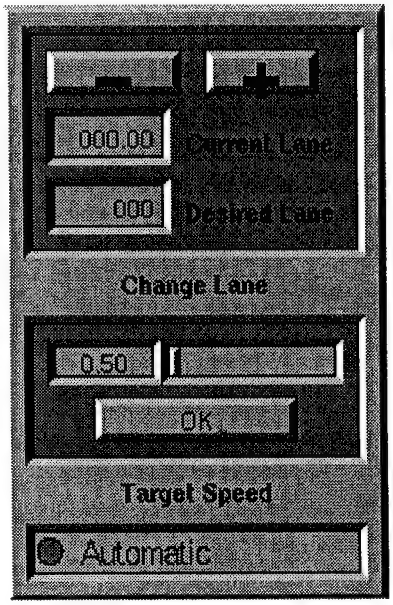


Figure 2: Control Panel for IM Mode

When a vehicle operates in *IM* mode, the user is allowed to make the selected vehicle change lane to right or left, one lane at a time, by pressing the + or – buttons. The selected vehicle then tries to perform the maneuver. It is possible that the vehicle is in the midst of a maneuver, when the user sends a change-lane request to the vehicle; in this case, the coordination layer of the vehicle will abort the maneuver it is involved in and, then, initiates the change lane maneuver. The user is also allowed to change the optimum velocity of the selected vehicle using the control panel. Speed changes are not instantaneous and occur under the control of the regulation layer feedback laws [6]. Also, note that a follower will not achieve its assigned optimum speed until it becomes a leader. *IM* mode allows testing of the different maneuvers and control laws proposed for an automated vehicle.

Manual (*Ma*) In the manual mode, the user controls all the functionalities of a vehicle. The animation module, as in the *IM* mode, provides the interface. With this interface, the user provides acceleration or deceleration and the steering angle for the vehicle, and effectively "drives" the car. The regulation layer of the vehicle calculates the longitudinal and lateral position and speed of the vehicle accordingly. With this mode, one can build different scenarios like a stopped car or an accident and observe their effects on the simulated traffic. Also, one can devise and test routines for the interaction between the automated and manual cars.

The three modes of operation described above are completely decoupled from each other; so, every mode has its own supervisor and a set of supporting modules that it requires. For example, the *Ma* mode doesn't need communication, sensors, and maneuver modules, but it needs the regulation layer for calculation of the car trajectories, *IM* mode doesn't need the car-to-link communication facility, but it has all other modules, and the *Au* mode has all the modules. Since it is possible to switch from one mode to another during the simulation (by selecting a vehicle and changing its mode through animation interface), we need to have a mechanism for switching from an old mode to a new one which we explain next.

SmartPath creates a car in the highway by initializing the *supervisor* module of the operational mode. A generalized state diagram for a supervisor module is shown in figure 3.

After initialization, the supervisor module activates the other modules which it needs. In the second state, it operates as a supervisor of the vehicle's maneuvers and activities. When the user decides to change the mode of the car, the animation module sets an event for the supervisor requesting the mode switching. This event causes the supervisor to move to its third state which is to prepare the vehicle for the next mode. The complexity of this state depends on the new mode as well as the current mode. Ther are no preparation tasks for mode switching from *Au* mode to *IM* mode and from *Ma* to either *IM* or *Au*, since the *Au* and *IM* modes have the same range of operation, and they both cover the *Ma* range. Switching from *Au* or *IM* mode to the *Ma* mode is fairly complex, since the *Ma* mode doesn't support platooning. Therefore, before the initialization of *Ma* supervisor, the vehicle has to become a free agent a one-car platoon[2]. Also, the supervisor has to process all its pending messages and send an appropriate reply to each one. If the free agent maneuver is not successful, there is a fault in the system and the simulation stops.

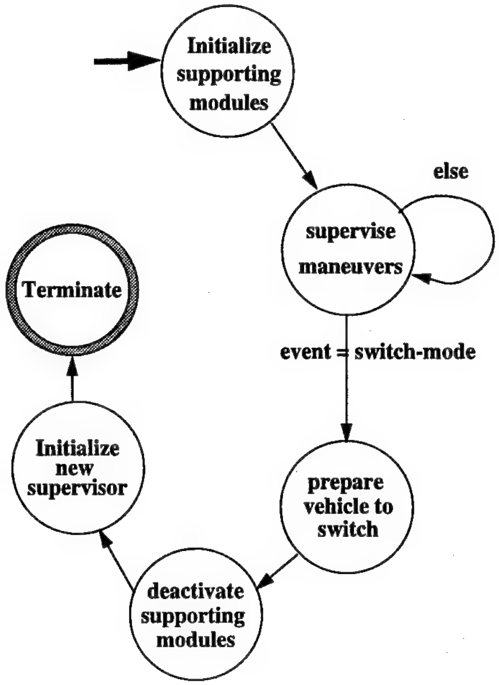

Initialize supporting modules
else
supervise maneuvers
Terminate
event = switch-mode
Initialize new supervisor
prepare vehicle to switch
deactivate supporting modules

Figure 3: Supervisor's State Diagram

After the current supervisor prepares the vehicle for mode switching, it deactivates its supporting modules, initializes the new supervisor, and terminates itself.

4 Conclusion

SmartPath at its present state is a simulation package for an AHS, and is directed toward the control hierarchy discribed in section 2. The interactive mode of *SmartPath* provides the ability to select a vehicle, change its mode of operation, and if desired, "drive"

[2] Free agent maneuver is described in [5]

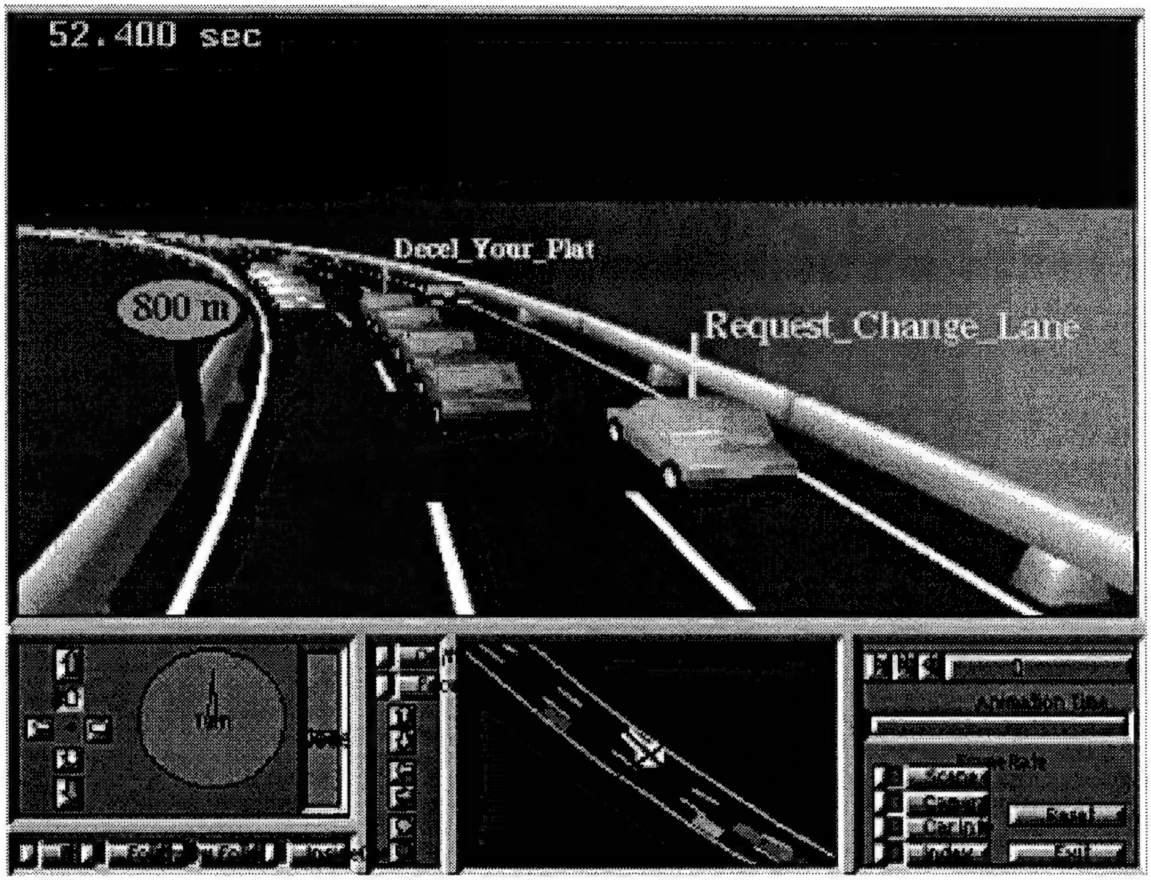


Figure 4: A Frame of Animation

the car. This produces much desired flexibility within the simulation, and allows the user to study and experiment with various control laws and vehicle models.

In its future revisions, *SmartPath* will be extended to include different control strategies which may be proposed for AHS.

Acknowledgement

We are grateful to Prof. Pravin Varaiya, Datta godbole, and John Lygeros for their comments and insights which improved the design of the interactive mode.

References

[1] B.S.Y. Rao and P. Varaiya, "Roadside intelligence for flow control in an IVHS," *Transportation Research Journal, part C*, vol. 2, no. 1, pp. 49–72, 1994.

[2] B.S.Y. Rao, P. Varaiya, and F. Eskafi, "Investigations into achievable capacity and stream stability with coordinated intelligent vehicles." Transportation Research Board, 1993.

[3] P. Varaiya and S. Shladover, "Sketch of an IVHS systems architecture," in *Proceedings of the Vehicle Navigation and Information Systems Conference*, (Dearborn, MI), pp. 909–922, October 20-23 1991.

[4] F. Eskafi, D. Khorramabadi, and P. Varaiya, "SmartPath: an Automated Highway System Simulator," tech. rep., PATH Technical Note UCB-ITS-94-4, Institute of Transportation Studies, University of California, Berkeley, CA 94720, 1994.

[5] A. Hsu, S. Sachs, F. Eskafi, and P. Varaiya, "The design of platoon maneuvers protocols for IVHS," tech. rep., UCB-ITS-PRR-91-6, Institute of Transporation Studies, University of California, Berkeley, CA 94720, April 1991.

[6] D. Godbole and J. Lygeros, "An interface between continuous and discrete-event controllers for vehicle automation," tech. rep., PATH Memorandom 93-8, Institute of Transportation Studies, University of California, Berkeley, CA 94720, August 1993.

Computing RF Propagation for Use in Simulation, Modeling, and Analysis

Scott Fehr, David A. McClung, and Greg Nagao
Applied Research Laboratories
The University of Texas at Austin

Abstract

The Radio Frequency Mission Planner (RFMP) provides analysis and simulation of communication links based on RF propagation models to more accurately assess performance and capability of transmitters and receivers for command and control warfare (C2W) missions. RFMP processing is dependent on equipment parameters, environmental factors, topography, and force structure. Two and three dimensional displays of results allow an operator to quickly evaluate RF propagation possibilities for determining the probability of maintaining connections.

1: Introduction

Communications connectivity and radio frequency (RF) wave propagation for warfare simulation have basically been ignored, or at best have been approximated with equipment planning ranges. It is now possible to provide analysis and simulation of communication links based on RF propagation models to more accurately assess performance and capability of transmitters and receivers for command and control warfare (C2W) missions. The Radio Frequency Mission Planner (RFMP) under development at the Applied Research Laboratories, the University of Texas at Austin (ARL:UT), can now provide simulation-based analysis of RF coverage and connectivity problems. RFMP processing is dependent on equipment parameters, environmental factors, topography, and force structure. Two and three dimensional displays of results allow an operator to quickly evaluate RF propagation possibilities for determining the probability of maintaining connections. RF analysis will also include the ability to view footprints and ground tracks of up to eight satellites. For geolocation, RFMP will provide an estimate of an error ellipse for a global positioning system (GPS) based time difference of arrival system. The operator will be able to visualize and understand how an 'optimum' geometry results in a minimum geolocation error.

Data sources. RFMP currently receives real-time and historical data through extensive connection to both external and local communication feeds and databases. External data sources are accessed via integration under the Unified Build component of the U.S. Navy Joint Maritime Command Information System (JMCIS). JMCIS provides a single integrated communications system for major Naval command centers and auxiliary commands. A central database server is integrated with an automated message handling system for rapid access to reference/tactical databases with technical and operational information on emitters, receivers, environment, and status of forces message validation and parsing. Local data sources will provide real-time parameters from GPS based Total Electron Content (TEC), Computerized Ionospheric Tomography (CIT), and environmental noise collection components which are under development at ARL:UT. Defense Mapping Agency digital terrain elevation data is used to provide path profiles for terrain dependent calculations. Environmental parameters are extracted from other JMCIS sources such as the Naval Integrated Tactical Environmental System.

For example, an RF reception mission requires spatially positioning one or more RF sensors to detect the emissions of a transmitter at a known position (with a possible associated geolocation error). Transmitter characteristics (frequency, bandwidth, modulation,...) and characteristics of the medium (atmosphere, terrain) through which the RF energy is to propagate must be known or approximated. Digital Terrain Elevation Data (DTED) provides terrain

ISBN 0-8186-6440-1. Copyright (C) 1994 IEEE. All rights reserved

0-8186-6440-1/94 $04.00 © 1994 IEEE

relief information. Environmental noise values are provided by either models or actual noise measurements. Propagation path loss models are then run to calculate the field strength at potential receiver locations. Threshold values are set to clearly illustrate good sensor locations on an electronic map. Plans are underway to include anomalous propagation modes such as transequatorial, sporadic-E, and meteor scatter.

2: RFMP Operational Areas

RFMP will run as a tactical decision aid under JMCIS and is being developed to serve primarily as an RF propagation planning, management, and analysis tool for the Naval Security Group. The mission types supported include: COMMS (communicate between own forces) and EVADE (avoid detection by non-friendly forces).

In an operational environment, RFMP supports four activities summarized here. ***Familiarization:*** familiarize the analyst with RF propagation in the expected area of operations and lead the operator to understand salient geographical and environmental properties as they relate to RF propagation. ***Planning:*** categorized into long-range, medium-range, and short-range mission planning. RFMP will identify candidate locations, platforms, and configurations required for the mission types, and will lead the analyst through the planning process. Future development includes the ability to recommend a locally optimum set of taskings based on command requirements and available resources. ***Management:*** monitor, control, and manage electronic assets during the course of an operation. ***Evaluation:*** during and after the conduct of the mission, assets provide reports back to the RFMP. The RFMP will evaluate performance and will provide possible explanations for deviation from predicted performance. To support a distributed training/planning situation like DISE, a variation of these activities may be more appropriate. In the remaining paragraphs of section two, we illustrate the application of RFMP to three important electronic missions.

COMMUNICATIONS: (Primarily for communications-electronics staff.)

Familiarization: Communications-electronics (CE) staffs familiarize themselves with propagation in the area of interest. Coverage for specific communications equipment is explored. If a commander is considering a specific location for a command post, the coverage to subordinate elements for tactical AM and FM radios is very quickly analyzed. Visual representation of HF propagation variations due to time of day or sunspot cycles is provided. Link coverage is also explored for microwave systems and backbone systems such as Mobile Subscriber Equipment.

Planning: Having become familiar with characteristics for a specific area, CE planners use RFMP to determine best locations for various types of connectivity. Given the simulated deployment and organic assets, net coverage for HF and VHF is displayed. Access areas for MSE Radio Access Units and connectivity along a route of attack is also shown. HF frequency assignments are verified for time of day and sunspot cycle, alerting planners to requirements for additional frequencies before the fact. The best HF antenna application for each station is verified. Transmitted power required to close each link is determined so that lower power levels are used where possible to minimize vulnerability to detection by opposing forces. Location adjustments are explored to use terrain blocking where feasible to also minimize vulnerability to detection and jamming. Finally, planners identify requirements for additional assets such as repeaters or tactical satellite communications equipment to ensure adequate communications capacity to units for successful command and control.

Evaluation: During the conduct of the simulation, the RFMP provides players and controllers with an analysis of the simulated nets helping players to better understand and apply proper ECCM techniques as well as actions that will optimize connectivity while minimizing vulnerability. Controllers use similar analyses to prevent the use of communications systems that would not be able to cover ranges or topography in a real tactical situation similar to the simulated deployment.

EVASION: (For operations, intelligence, and CE staff)

Familiarization: RFMP allows all fighters to explore options of visual and electronic cover and concealment. Evasion is particularly critical to reconnaissance units, special warfare units, and aircraft routes deep into enemy territory. Based on the estimate of the situation, visual and electronic coverage of the battlefield is depicted. Routes and locations are then identified which minimize detection.

Planning: In the planning phase, all staff members are concerned with visual and electronic cover and concealment of movements and locations. Based on the estimate of the enemy situation, realistic coverage of sensors and observation posts are depicted to aid in the selection into and out of opponent areas. Further, checkpoints are identified along the routes that provide communications connectivity to command authorities while offering the maximum protection against detection. (The identification of key terrain such as a mountain to shield emanations from opposing sensors is just one such example.)

Evaluation: Evaluation of evasive tactics is a controller function. The controllers use RFMP to analyze decisions and actions of commanders and staffs as well as movement of units to determine when actual detection or direction finding is likely to occur by opposing forces.

Such analysis adds realism to the simulation as controllers can point to specific opposing units or assets that made the detection. Controllers are also able to more authentically illustrate actions that may have avoided detection.

3: RFMP Functional Description

Each type of RF propagation analysis performed by RFMP can be described in terms of a hierarchy of analysis functions which serves a dual purpose. Traversed from highest-to-lowest level functionality, this hierarchy generates a requirements specification for the analysis in which raw data input requirements are at the lowest/terminal nodes of the hierarchy. Traversed from lowest-to-highest level functionality, the hierarchy generates a data flow model for the computation.

The hierarchy naturally decomposes into two parts: 1) RF propagation - an analysis of how the electromagnetic radiation propagates from a source to various locations through interactions with the environment; and 2) Signal reception - an analysis of a received signal for the purpose of extracting information about its source (direction-finding) and/or its content (signal analysis). Propagation path loss models, such as TIREM [1], Advanced PROPHET [2], FFACTR [3], and RPO [4], are the core functions within the RF propagation subsystem. These models provide path loss and field strength estimates from 3 MHz to 20 GHz over a geographical volume of interest which may include land and water.

As an example, consider the use of the propagation models to calculate a mission analysis called the "probability of detection". Starting with the power of the transmitter PT, we use the propagation models to calculate the propagation loss PL, use PL to calculate the field strength FS, integrate FS over the receiver antenna configuration to give the signal strength SS, calculate the received signal power SP, and signal power level SPL, and finally evaluate the results through the probability of detection POI. Diagrammatically, we have

PT --> PL --> FS -->||-->SS -->SP --> SPL --> POI

where the double bar || indicates the division between the propagation and signal reception analysis. The RFMP uses HF, VHF, and UHF propagation models to help perform the RF propagation analysis. The main modules of these propagation models evaluate the effect of environmental interactions with RF electromagnetic propagation. The various outputs of these models, such as propagation loss PL may generally be used to calculate FS associated with the transmission due to a give source. Once the RF propagation analysis has been performed, further analysis can be performed on the data to help determine where receivers of different characteristics (such as antennas, sensitivity, signal-to-noise ratios, etc.) are most likely to "hear" the given transmitter. This information may be evaluated in the context of the probability of detection function.

The above analysis provides strong constraints on the interface between the propagation models, environmental data, and the analysis types. The mathematical formulation of the constraints on the interfaces provides a strong basis in which to evaluate the interplay between the various subsystems which compose the RFMP. In particular, the interfaces to the propagation model outputs can be mathematically specified based on its role in the analysis, and the model input parameters requirements can in turn be mathematically specified. Further, based on the mathematical formulation of constraints on model parameters, the effect of environmental parameter uncertainties on the RFMP probability analysis can be determined.

4: Evaluation of RF Mission Success

RF analysis is combined with noise levels from models or actual measurements and interference on frequencies of interest to provide a statistically based estimate, or probability, of RF mission success. This feature allows the operator to visualize the stochastic nature of RF propagation modeling results and provides an estimate of how changes in environmental conditions and geometry affect the probability of mission success. RFMP will correlate predictions to actual performance and will provide explanations for possible causes of differences. Recommendations to improve performance, such as repositioning assets, will be provided. The evaluation process may also reinforce the conclusion that for a given set of circumstances, accomplishment of the mission is simply not possible.

5: Adapting RFMP for DIS

As stated, the RFMP is connected to both local and external (via integration under JMCIS) communication feeds and databases. In this configuration, RFMP acquires data needed for RF problem simulation directly (under its control) via the JMCIS Client-Server Architecture, using APIs for database servers (e.g. CDBS), or from broadcast sources such as JMCIS messages.

Adapting RFMP to DIS will require redirecting its external API model to rely on other DIS server objects as sources for data, replacing local JMCIS APIs. The DIS interface will replace the JMCIS Comms module as the source of tactical message traffic and database updates. The combination of operator/analyst inputs and RFMP database queries will no longer be required to specify the current problem; instead, DIS servers will provide data as part of Ground Truth and Update broadcasts for the current simulation. Under DIS, the role of the operator/analyst will be to monitor the RFMP concept of the game, and interpret RFMP results based on graphics and textual analysis, then construct the RFMP analysis PDU and send to DIS.

The JMCIS Chart Server would be replaced by the new DIS block format to transfer DTED terrain data. The DIS API will differ in form from the JMCIS Client-Server Architecture, but data element formats within DIS PDUs should be the same as from JMCIS databases, since DIS and JMCIS are moving toward a common standard.

This paper presents an available austere swivel-chair type interface between RFMP and DIS. A more desirable, automated interface would allow RF analyses to govern transmittal of messages even between commands of units played totally within the simulation thus eliminating the need for an umpire to censor unlikely connectivity. The specification of at least three PDUs for such an interface is an open issue. One PDU would be necessary to pass the analysis requirement and specification to RFMP. Another is required for the high resolution measurement data needed for RFMP probability based analysis. A third would return results to DIS. The RFMP team should play and active role in this PDU definition, and in the specification of an RFMP output PDU to return the results of RFMP analysis to DIS.

6: Conclusion

Almost since its inception, the Signal Corps has professed that when others go to the field to train, signal soldiers go to the field to do their job. While tacticians plan in potential hostile regions, communicators and IEW units install, operate, and maintain assets as deployed rather than as simulated. A tool is available today to allow simulation of RF challenges faced by planners and operators of the electronic battlefield. RFMP will enhance the synthetic environment to identify assets to overcome electronic and propagation obstacles much the same as transportation and engineer assets are now identified to overcome physical obstacles. An austere, yet sophisticated, interface can be realized immediately to provide commands and staffs with a realistic representation of the RF environment and its impact on command, control, and communications.

7: Acknowledgment

The authors thank the Naval Security Group for their support of this project under Contract N00039-91-C-0082-6-81-1.

8: References

[1] TIREM/SEM Handbook, Report # ECAC-HDBK-93-076, by Epplink, D. and Kuebler, W., IIT Research Institute, March 1994.

[2] Operational Users Manual for Advanced Prophet Version 4, Naval Command, Control, & Ocean Surveillance Center, RDT&E Division, Signals Exploitation Branch, August 1985.

[3] Engineer's Refractive Effects Prediction System (EREPS) Revision 2.0, Report # NOSC TD-1342, by Patterson, W.L., et al, Naval Ocean Systems Center, February 1990.

[4] Radio Physical Optic CSCI Software Documents, Report #: TD-2403, by Patterson, W.L. and Hitney, H.V., Naval Command, Control and Ocean Surveillance Center, R&D, T&E Division, December 1992.

Session 2G:

Applications II

Integration of CGF with Fielded Equipment Using DIS

Phil Landweer
BDM Federal

Abstract

A unique application of a Computer Generated Force (CGF) was conducted in December of 1993 at the Depth and Simultaneous Attack Battle Lab in Ft. Sill, Oklahoma. The CGF simulated a fire support scenario with tanks, infantry fighting vehicles, artillery units, counter-battery radars, and associated command and control elements. A tactical situation display showed the locations of all combatants, as well as activities of interest such as detections, weapon firings, detonations, and communications as the simulated battle progressed in real-time. A DIS-compliant interface allowed the CGF to interact with actual fire support equipment, both sending and receiving PDUs to a variety of systems. These systems included a DMD at the simulated Fire Support Element, FEDs at the forward observer and fire support team, LCUs serving as an FDS or FDDM interface for the FDC and MLRS Battalion, respectively, and an MLRS Fire Control Panel Trainer. Thus, a seamless simulation was provided between constructive, virtual, and live simulations.

1. Application overview

This project was conducted at the Depth and Simultaneous Attack Battle Lab (D&SA BL). The U.S. Army uses its Battle Labs to quickly investigate the utility of candidate systems, architectures, and tactics. The D&SA BL focuses on those systems which can be used for attacking the enemy from long distances or in a coordinated fashion, such as fire support systems.

The purpose of this project was two-fold. First, the Army Research Laboratory (ARL) Ft. Sill Field Element, which sponsored this effort, wanted to investigate how mission performance improved for beginning Artillery School students. Each student used a Multiple Launch Rocket System (MLRS) Fire Control Panel Trainer (FCPT) to execute Call for Fire missions within a simulated battle. Each student participated in the battle three times, and the timeliness with which the fire mission was executed was measured. The other purpose of the project was to determine how well Distributed Interactive Simulation (DIS) Protocol Data Units (PDUs) could be used to integrate fielded equipment with each other as well as with constructive and virtual simulations.

2. CGF description

The CGF system used was CIMUL8^{TM}/SPECT8^{TM}/DISIP8, a commercial off-the-shelf (COTS) software product. CIMUL8 is the simulation engine, and models tactically representative behaviors based upon user inputs. CIMUL8 also has a self-contained pre-processor for building up units and scenarios, as well as a post-processor for analyzing battle outcomes. SPECT8 is a graphical display system, and may be used to preview, replay, or watch a CIMUL8 run as it progresses. Finally, DISIP8 is a DIS-compliant interface used to both send and receive PDUs between CIMUL8 and other assets. SPECT8 can also be used to display the occurrence and effects of received PDUs.

3. Simulation characteristics

CIMUL8 models both physical phenomena and cognitive processes. This combination allows all pertinent areas of a battle to be simulated. An object-oriented approach is used, wherein real-world entities are modeled as coherent collections of physical and cognitive objects.

ISBN 0-8186-6440-1.

0-8186-6440-1/94 $04.00 © 1994 IEEE

3.1 Physical modeling

The six object classes within the physical domain are Sense, Talk, Move, Shoot, Disrupt, and Replenish. Sense models the generalized concept of non-cooperative information exchange. Non-cooperative means that the sensed object is not doing anything intentional in sending information to the sensor. Radars, thermal tracking devices, and the human eye are all examples of Sense objects. Talk, on the other hand, models cooperative information exchange. Radios, modems, and digital SatCom links would all be modeled using Talk objects. Move allows a simulated unit to change its location within the simuland. This includes driving across the surface, flying through the air, and navigating the oceans. Shoot models the generalized concept of lethal engagements wherein the actor is attempting to physically harm the target. Bullets, shells, and directed energy devices are examples of Shoot objects. Disrupt is the converse of Shoot in that it is used to model non-lethal engagements of other players. The intent of a Disrupt object is to deny some other entity the use of its Sense, Talk, or other objects. Smoke, chaff, and active radio jammers are examples of Disrupt objects. Note that using a Disrupt object against cognitive objects allows for psychological operations to be simulated. Finally, a Replenish object models the generalized concept of material exchange. This includes repairing damaged objects, resupplying ordnance or fuel to maneuver units, and performing routine maintenance.

3.2 Cognitive modeling

The advantage of using cognitive objects to represent behaviors within the simulation is that it decouples "reality" within the simulated battlespace from the "perceived reality" that is used by the personnel and units within the simuland. Without this decoupling, the modeled entities would directly use ground truth, and thus conduct themselves as though they were omniscient. Of course, real-world combat as well as exercises don't work this way. At the D&SA BL, this feature of CIMUL8 allowed the students to be placed in a very realistic environment. There are also six cognitive objects within CIMUL8: Notice, Digest, React, Review, Plan, and Adapt.

A notice object models the perceptual recognition of a stimulus. This includes noticing the receipt of a message over a communications device, noticing that an OPFOR unit has just come into view from behind a building, or noticing the effects of ordnance either on oneself or against an enemy. The notice objects are explicitly linked to the provider of the stimulus within CIMUL8. This adds an extra degree of realism onto the simulation, and prevents an object from noticing something that it wouldn't have access to. A notice object moves an image from iconic memory into short term memory. Once there, a digest object will operate on the data.

A digest object models the correlation, fusion, and assimilation of new information with information already on hand. Thus, digest takes the data content of an item from short term memory and moves it into medium term memory. Medium term memory is where the current perception of the battlefield is held for each unit within the simulation. This includes the perceived location of both the enemy's and one's own units, the status of organic and attached units, and the current perceived intent of the enemy. As new data is synthesized into the present body of knowledge for a unit, incorrect perceptions may be formed in a number of ways. The incoming stimulus that was noticed may have errors, the filters may produce incorrect deductions, or the uncertainties may be mitigated in the "wrong" way. And if the perception of the battlefield in medium term memory is incorrect, then mistakes will be manifested by a react object, as discussed below.

A review object is somewhat of the converse of the digest object, in that it is responsible for managing the "length" and "veracity" of medium term memory. A review object performs the functions of discarding information that is out of date, "known" to be incorrect, or no longer needed due to changed conditions. Review does this by evaluating the quality of information over time using moving averages of an n-dimensional perception space, where n is specifiable by the user. Based on the evaluation, a review object may throw away perceptions, draw new conclusions, or request additional information. Thus, review will either delete items from medium term memory, or cause a series of events that (hopefully) results in new stimuli to be noticed.

A react object evaluates the present perception of the battlefield using the present contents of medium term memory, makes a decision, and then may do something based on the decision made. The decision space is countably infinite, using geometrical, status, intent, and relational parameters built up into rule sets of variable resolution. The rule sets are used to dictate the behaviors of units for movement, command and control decisions, weapons usage, countermeasure employment, resupply and repair, and system status changes. Since combat activities which result from these decisions are based upon the perceptual view of the battlefield, incorrect perceptions may generate mistakes in a simulated unit's behavior.

A plan object measures a unit's progress towards its goals and assesses the need for changed strategy. The goals along with candidate plans are stored in long term memory within the simulation. The present plan and goal(s) are stored in medium term memory. Combining these with the perceptions which are also in medium term memory, progress is measured along some user-specified n-dimensional metric. The same criteria used by a react object are also available to a plan object for measuring progress. As a unit's plan object decides to change a current plan, the current plan in medium term memory is replaced by a plan from long term memory.

Finally, an adapt object can modify a plan (or "strategy") that exists within long term memory. The adapt object can thus alter the way an entity evaluates and reacts to situations. These objects allow entities to "learn" from simulated combat experience. Again, measurable criteria from those used by the react object are used by an adapt object to assess whether or not a given combat outcome is "good" or "bad".

3.3 Event Scheduling

Within CIMUL8, both physical and cognitive events are scheduled dynamically as the battle progresses. This provides a natural means of simulating battlefield activity, since any combatant can be represented as some collection of the physical and cognitive objects described above. In addition, the event driven nature of CIMUL8 allows for other simulations (either constructive, virtual, or live) to be easily integrated into the overall scenario by injecting their events into the simulation.

Figure 1 shows how the physical and cognitive objects interact with one another to simulate the behavior of an entity, such as an MLRS launcher. The physical objects are shown in white, with the cognitive objects shaded. Starting near the center at the top of the figure, the MLRS might get a Call for Fire message from the FDC, which is its commander. A *Notice* object will notice the receipt of the message, which would then get processed by a *Digest* object to discern the message content. If the message were a move order, then the MLRS might decide to start movement using its *Move* object, and continue to move until the objective was reached. Note that this movement might cause the MLRS to come into view of other entities' sensors, which would give them the chance to see the MLRS. This is depicted by the thick arrow labeled "to other Sensors." However, since the message was a Call

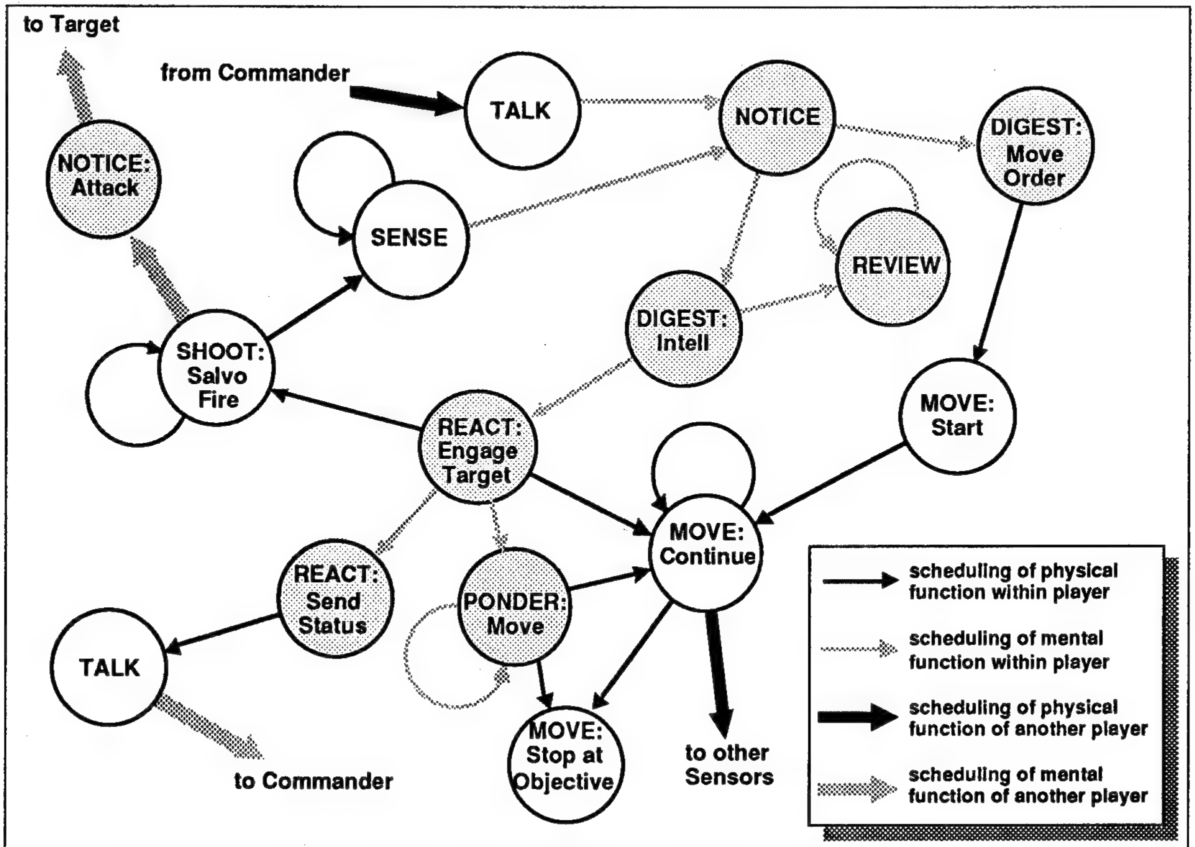


Figure 1: Even Scheduling between Physical and Cognitive Objects

for Fire, the MLRS' *React* object will be schedule to consider engaging the target. Should the MLRS do so, the *React* object will have the MLRS report its status back to the FDC, using a *Talk* object to model the communication. Upon the decision to fire, a *Shoot* object will simulate a weapon firing at the target. Our example ends with the target's *Notice* object noticing that it is under attack!

4. Scenario descriptions

A ground combat scenario was simulated for this project. The "Blue" Forces (BLUFOR) consisted of an M1 tank platoon accompanied by an M2 Infantry Fighting Vehicle and two Improved TOW Vehicles (ITVs). Fire Support units including a Forward Observer (FO), Fire Support Team (FIST), Fire Direction Center (FDC), Fire Support Element (FSE) with a TPQ-36 fire-finder radar, MLRS Battalion, and MLRS Self-Propelled Loader Launcher (SPLL) were in direct support. The Opposing Force (OPFOR) had a Motorized Rifle Company with BMPs and T-80 tanks, a Self-Propelled Howitzer (SPH) Battery, and associated FDC.

Depending on the particular configuration used, these combatants were simulated using some combination of CIMUL8 (a constructive simulation), the FCPT (a virtual simulation), and fielded equipment (live simulations). The fielded equipment included Field Entry Devices (FEDs), a Digital Message Device (DMD), Low-cost Computer Units (LCUs), and a Fire Direction Data Manager (FDDM). For each situation, a configuration file instructed CIMUL8 how the simulated activity was to be distributed across these domains.

One interesting facet of this project was discovered upon arriving at the D&SA BL three days before the first demonstration was to occur. Initially, the scenario within CIMUL8 was located within northern Europe. Unfortunately, the FCPT could only execute fire missions using coordinates from the Ft. Sill firing range. So, the entire simulated battle was "moved" from a European battlefield to the Ft. Sill area. This translation was accomplished in a single afternoon, and included repositioning all combatants into realistic positions and ensuring that the required interactions would occur for the project. Figure 2 shows the relative

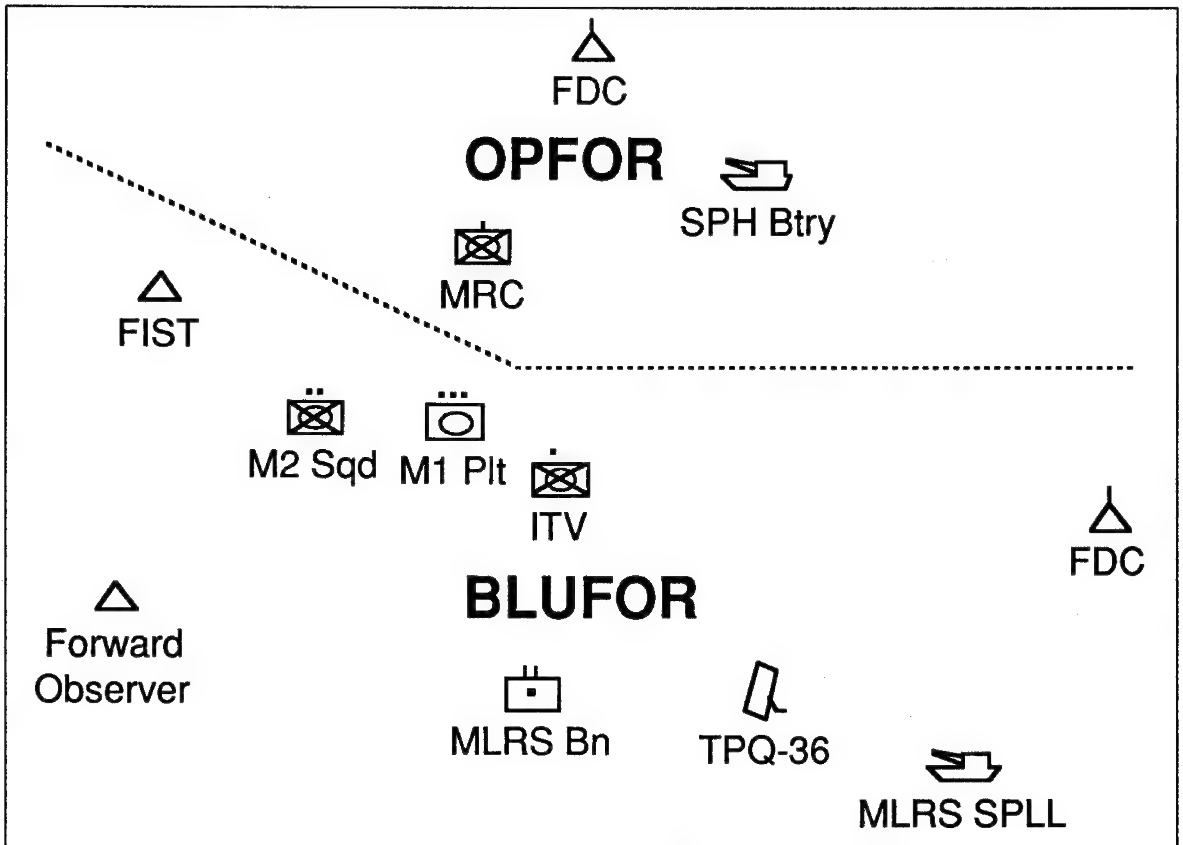


Figure 2: Laydown of the Simulated Forces

from the north, with the BLUFOR in a defensive posture.

5. DIS protocols used

DIS 2.0.3 PDUs were used to interface the simulations with each other. Specifically, Entity State, Firing, Detonation, Transmit, and Signal PDUs were used. Transmit and Signal PDUs were used to send TACFIRE messages between the various assets. A PC-based DIS interface developed by CAE-Link was used to transform tactical communications into DIS PDUs and vice versa. In this manner, the fielded equipment performed just as it would in a real combat environment.

6. Asset configurations

Four different configurations of the basic ground combat scenario were used at the D&SA BL. Each of these will be described in turn. First, the aspects which are common to each scenario will be described.

6.0 Common aspects

CIMUL8 simulated the movement, signatures, sensing, command and control, communications, engagements, firing, and lethality effects of all OPFOR units as well as the BLUFOR tanks, IFV, and ITVs. The movement of the vehicles followed pre-planned maneuver profiles specified by the scenario designers. Signatures included both the normal visual cross section of the vehicles, as well as increased signatures in the visual and IR spectra whenever a round of ammunition was fired. In turn, the visual and thermal sensing systems would allow the simulated combatants to detect the apparent position of the vehicles. These detections would lead to perceptions of the battlefield being formed, and used for C^3 or engagement purposes. The command and control, engagement, and firing procedures were specified via rule sets to CIMUL8. These rule sets make up a context-sensitive language that allows the simulated entities to behave in very realistic, user-specified ways. Should a situation arise requiring a commander to direct a subordinate in the battle, tactical communications were simulated as VHF radios with point-to-point signal transmissions. Finally, battlefield effects were modeled using direct- or indirect-fire lethality assessment tables.

Whenever Transmit and Signal PDUs were generated by the live or virtual simulations, SPECT8 would display these events as green star bursts around the transmitting unit. This allowed everyone to watch the information flows as the battle progressed.

6.1 First situation: FO, FIST, FDC, and FCPT

The FO's viewer was modeled within CIMUL8. As the OPFOR targets came into view, the targets were visually acquired within the simulation. SPECT8 was used to display this event by placing a large diamond around the acquired targets on the tactical situation display. This served as the triggering event for an operator to enter the target coordinates into the FO FED and send an FR Grid TACFIRE message to the FIST FED. This message was actually transmitted as a set of Transmit and Signal PDUs via Ethernet. Upon receipt of the message, the FIST operator would then pass the target coordinates on to the FDC. Another TACFIRE message was sent from the FIST FED to the FDC LCU, which was configured as a Fire Direction System (FDS). The FDC operator then generated a fire mission for the MLRS, and sent a Call for Fire message to the FCPT. The MLRS operator would then execute the mission, and fire upon the OPFOR within the simulated environment. Fire and Detonation PDUs were generated by the FCPT, which would damage or destroy the targets within CIMUL8.

6.2 Second situation: DMD, FDDM, FDC, and constructive MLRS

In this set-up, CIMUL8 was again used to start things off. A TPQ-36 fire finder radar was modeled, and could detect the OPFOR howitzers firing. As this happened, a large diamond was displayed around the targets on the SPECT8 screen. The target coordinates were then entered into the DMD by an operator, and a TACFIRE message sent via Transmit and Signal PDUs to the MLRS Battalion's FDDM. Upon receipt of the target by it's LCU, the Battalion would then task the FDC with a fire mission. TACFIRE messages were sent between the LCUs. Upon the FDC sending a Call for Fire message to the MLRS, SPECT8 would display this message transmission. This cued an operator to use SPECT8's "Personal Control" capability to execute the fire mission. Thus, a constructive simulation was triggered by fielded equipment to complete the mission. Upon firing, the simulated rockets went to the targets and damaged or destroyed some of the OPFOR howitzers.

6.3 Third situation: FO, FIST, FDDM, FDC, and FCPT

This case was essentially a combination of the previous two. CIMUL8 modeled the acquisition of the OPFOR by the FO, with TACFIRE then being sent from the FO FED to the FIST FED. Then, the FIST would pass on the target to the FDDM. The FDDM would task the FDC, which then passed on the Call for Fire message to the MLRS FCPT. The operator would then execute the mission, which would damage or destroy the OPFOR vehicles.

6.4 Fourth situation: constructive FR grid, FDC, and FCPT

This was the set-up used for the actual ARL experiments. Figure 3 shows this configuration, with the laydown of the forces given using the same icons as in Figure 2. The quartered circles indicate whether or not the modeled entities can move, shoot/assign, sense, and/or communicate. The shading within each quarter indicates whether the simulated function was performed via constructive, virtual, or live simulation. The FO and FIST were wholly modeled within CIMUL8, including their acquisition and communication activities. After the FIST received the target, an FR Grid TACFIRE message was generated by CIMUL8, with DISIP8 sending out Transmit and Signal PDUs. This is shown in the figure using the solid arrow labeled "FR Grid." The FDC would receive these, with the operator then sending a Call for Fire to the MLRS FCPT. The student would then execute the mission, with the results of the firing effecting the overall battle outcome. All relevant data was captured within CIMUL8. This allowed CIMUL8's post-processor to measure the timeliness of the students' actions for quantitative analysis. SPECT8 could also be used immediately after each trial to replay the battle, thereby providing immediate feedback to the student.

7. Future applications

This seamless simulation technology may now be used for a variety of purposes. With additional FEDs, LCUs, and FCPTs, a large training exercise could be conducted within the U.S. Army Field Artillery School. Another potential application would be to integrate National Guard and Army Reserve MLRS units with an FDDM. The Guardsmen and Reservists could train at their home locations, with the overall simulation and FDDM located at Ft. Sill. This is particularly important for realistic training, since MLRS units would receive fire missions from an FDDM during

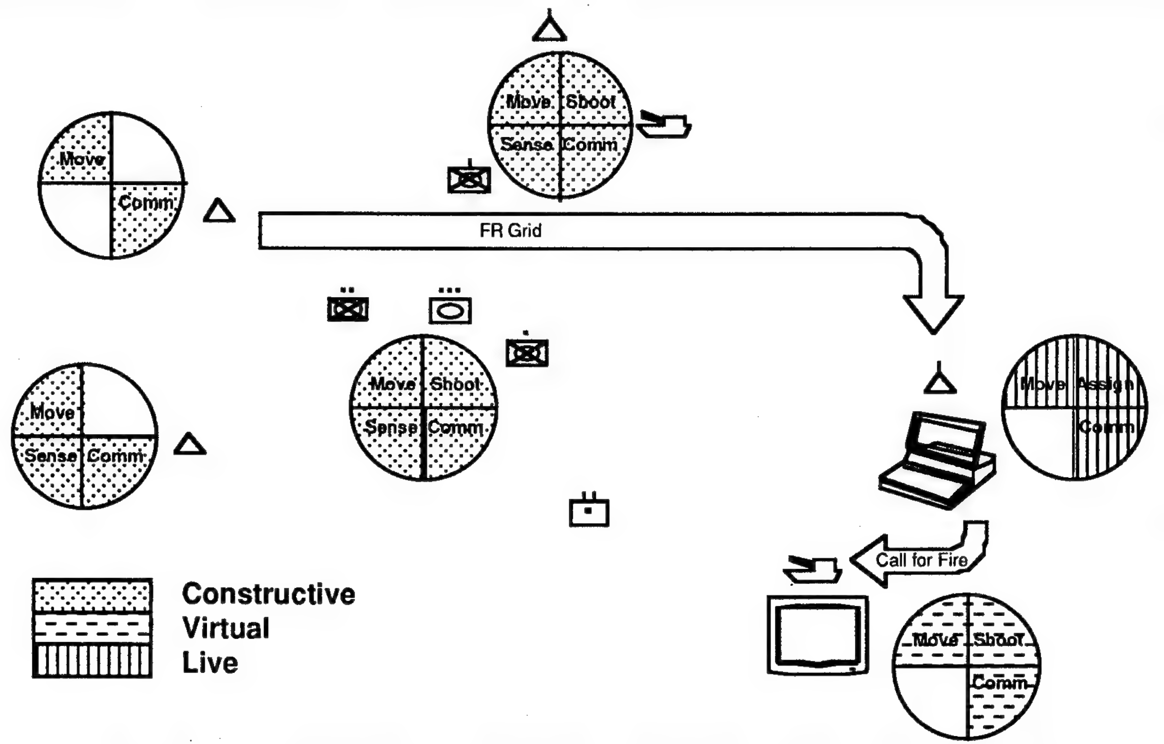


Figure 3: Distribution of Modeled Behaviors Used in Situation #4

combat, but Guard and Reserve MLRS units don't have FDDMs as part of their equipment. Finally, T-NET could be used as an enabling technology to link geographically distant equipment with each other, with DIS PDUs being passed from one location to the other via commercial satellite communications. Such technology could directly support Distance Learning projects.

8. Lessons learned

As the integrated simulation was used by the MLRS students, their proficiency in executing the Fire Mission improved. Such a result is to be expected. It was reassuring to see that DIS can be used to provide positive feedback and achieve training objectives, even when "high tech" systems are used. Probably the most challenging part of this project was getting all of the TACFIRE messages right so that the fielded equipment would operate properly. The Signal PDU is sufficiently flexible to contain any type of TACFIRE message. However, certain bytes of the FR Grid and Call for Fire formats are very critical! Also, the timing of TACFIRE message transmissions and acknowledgments must be closely adhered to if live simulations are to be integrated with each other and constructive and virtual simulations.

When using distributed simulation, one never knows which modeled units and associated behaviors should be simulated by constructive simulations, virtual simulators, or live simulation. The four situations described above allowed for a diverse assortment of virtual battlefields to be simulated by a mix of these three simulation domains. To meet the needs of the training or analytical effort being supported by simulation, the underlying architecture needs to be sufficiently flexible and reconfigurable. The success of this effort at the D&SA BL was due in part to this flexibility.

Biography

Phil Landweer works for BDM Federal, Inc. as the manager for Advanced Simulation. He has used digital modeling and distributed simulation for concept exploration, requirements analysis, pre-test analysis, test planning, test and evaluation, and tactics development to support a variety of government and industry organizations. His specific areas of interest include object-oriented simulation, functional modeling, and integrated computer graphics. Prior to joining BDM, he was an analyst at the Air Force Operational Test and Evaluation Center, where he used modeling and simulation to support a variety of Operational Test and Evaluation programs.

Fuzzy Finite Automata And Their Application To Speech Recognition

T. Van Le

Faculty of Information Sciences & Engineering,
University of Canberra,
PO Box 1, Belconnen, ACT 2616, Australia.

Abstract

A fuzzy pattern matching machine for speech recognition is simulated. Monosounds extracted from a set of keywords are stored in a fuzzy nondeterministic finite automaton, and a fuzzy pattern matching procedure is employed to activate the automaton for detection of the predetermined keywords in a given speech.

1 Introduction

The classical Aho-Corasick pattern matching machine (in [1]) detects predetermined keywords in a text by using a deterministic finite automaton supported by a backtracking procedure (which is called the *failure function* in [1]). In [3], I presented a method of implementing the Aho-Corasick matching machine as a nondeterministic finite automaton, using Prolog's automatic backtracking mechanism.

The recognition of keywords in a speech is more difficult, however, because sounds normally do not perfectly match like characters in a text string. Therefore, it is necessary to employ some fuzzy matching procedure to perform the approximate matching of two similar sound waves.

In this paper, I present a fuzzy pattern matching machine that follows the idea of Aho-Corasick [1] and that employs the technique of fuzzy pattern matching in [4] to detect predefined keywords in a speech.

The paper has the following sections. Section 2 describes the construction of nondeterministic finite automata to store the monosounds extracted from a set of given keywords. Section 3 presents the fuzzy pattern matching machine that will be used to activate the automata for detection of the keywords. The last section, Section 4, describes some experiments on a simulated pattern matching machine and presents the simulation results.

2 Fuzzy finite automata of monosound waves

Consider a set of keywords $\{w_1, \dots, w_n\}$, in which each word w_i is composed of a sequence of monosounds $x_{i1}, \dots, x_{ik_i}$. In general, a monosound can be decomposed into a finite number of sine waves, and is represented by a matrix of wave amplitudes. For convenience, let $\phi(s)$ denote the set of input signals (which are, in our case, the matrices representing the monosounds) that will transform the machine from state s to another state, and for each $x \in \phi(s)$, let $next(s, x)$ denote the set of states tranformed from s by the input signal x. Also, let $\tau(x, y)$ denote the matching degree of two monosounds x and y.

The following algorithm establishes a fuzzy nondeterministic finite automaton for the purpose of detecting the given keywords $\{w_1, \dots, w_n\}$ in a speech. Here, α represents a predetermined threshold for matching degree. Also, if there is a sequence of state transmissions

$$0 \xrightarrow{x_1} s_1 \xrightarrow{x_2} s_2 \xrightarrow{x_3} \dots \xrightarrow{x_k} s_k$$

then the sequence $x_1, \dots, x_k$ is called a route from the initial state 0 to the state s_k, and is denoted by $route(s_k)$. Note from the following algorithm that, initially, each set $next(s, x)$ contains at most one state, and each $route(s)$ is unique.

0-8186-6440-1/94 $04.00 © 1994 IEEE

Algorithm 1:

```
Set up the initial state 0;
Let $\phi(0) = []$ and $m = 1$;
For $i$ from 1 to $n$ do
   Start with state $s = 0$;
   For $j$ from 1 to $k_i$ do
      If $\phi(s) \neq []$ and $\max_{z \in \phi(s)} \tau(x_{ij}, z) = \beta > \alpha$
         then find $y$ in $\phi(s)$ such that $\tau(x_{ij}, y) = \beta$;
                 and let $[s] = next(s, y)$
         else  let $\phi(s) = \phi(s) \cup [x_{ij}]$;
                 $next(s, x_{ij}) = [m]$;
                 $\phi(m) = []$;
                 $s = m$;
                 $m = m + 1$;
   endfor;
endfor;
For each state $s$ such that $\phi(s) \neq []$ do
   For each $x \in \phi(s)$ and each $s' \in next(s, x)$ do
      Find the set $F$ of all states $s_0$ such that
         $route(s_0)$ is a suffix of $route(s')$;
      If $F \neq []$
         then let $next(s, x) = next(s, x) \cup F$
         else  let $next(s, x) = next(s, x) \cup [0]$;
   endfor;
endfor;
```

The output of Algorithm 1 is a fuzzy nondeterministic finite automaton that stores the monosounds extracted from the keywords $w_1, \ldots, w_n$.

3 The fuzzy pattern matching machine for speech recognition

The fuzzy finite automaton described in Section 2 is used as a knowledge base for our fuzzy pattern matching machine. The purpose of the machine is to detect the predetermined keywords (stored in the automaton) in a speech, which is represented as a sequence of monosounds. Here, the matching degree of two monosounds x and y (which are represented by two matrices) is defined by

$$\tau(x, y) = e^{-\|x-y\|}.$$

Two monosounds x and y are regarded as similar if $\tau(x, y) > \alpha$, where α is a predetermined threshold for matching degree. At any state s of the machine, if the set $keyword(s)$ of the keywords occuring in $route(s)$ is nonempty, then the detected keywords are recorded. The activation of the machine is expressed in the following algorithm.

Algorithm 2:

```
Set the initial state $s = 0$ and index $i = 1$;
Repeat
   Get the next monosound $x$;
   Find $y \in \phi(s)$ such that $\tau(x, y) = \max_{z \in \phi(s)} \tau(x, z)$;
   If $\tau(x, y) > \alpha$ (the threshold)
      then select an $s' \in next(s, y)$ and let $s = s'$;
            If $keyword(s) \neq []$
               then store the keywords and
                     their location $i$;
      else  backtrack to the previous selection;
   Let $i = i + 1$;
Until end of speech;
Ouput the stored keywords and their detected
locations.
```

4 Simulation results

A fuzzy nondeterministic finite automaton is established to store the keywords *fire, firebomb, fighter*, and *bonfire*. The automaton is depicted in the diagram of Figure 1.

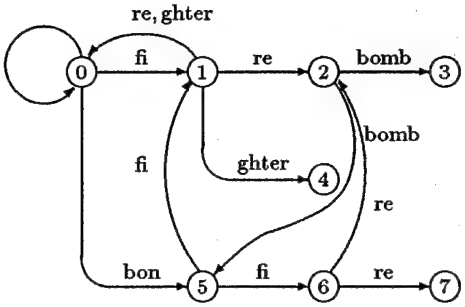


Figure 1

The soundwaves of the given keywords are shown in Figure 2. From these waveforms, monosounds are extracted by using a sound editor, and various bandpass filters are used to reduce the monosounds' frequencies to specific ranges before they are converted into frequency spectra (some of which are shown in Figure 3) by using the fast Fourier transform. The Fourier transforms of each monosound are stored as a matrix of real values. Algorithm 1 is then executed to generate a nondeterministic finite automaton that stores the monosounds, which are linked to their matricial representations in the knowledge base.

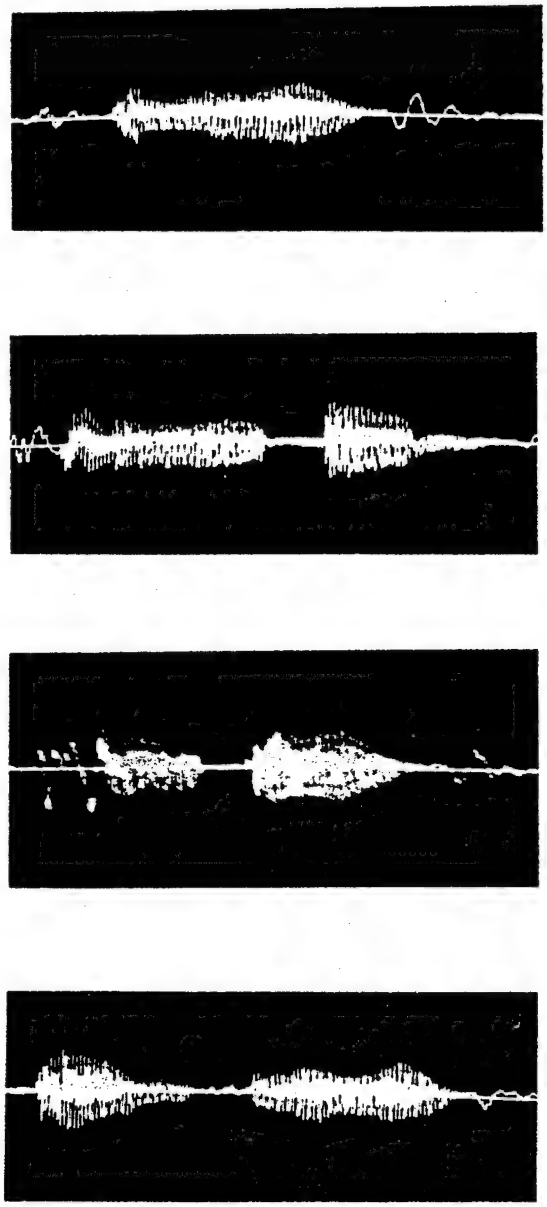


Figure 2: Waveforms of the words *fire*, *firebomb*, *fighter*, *bonfire*.

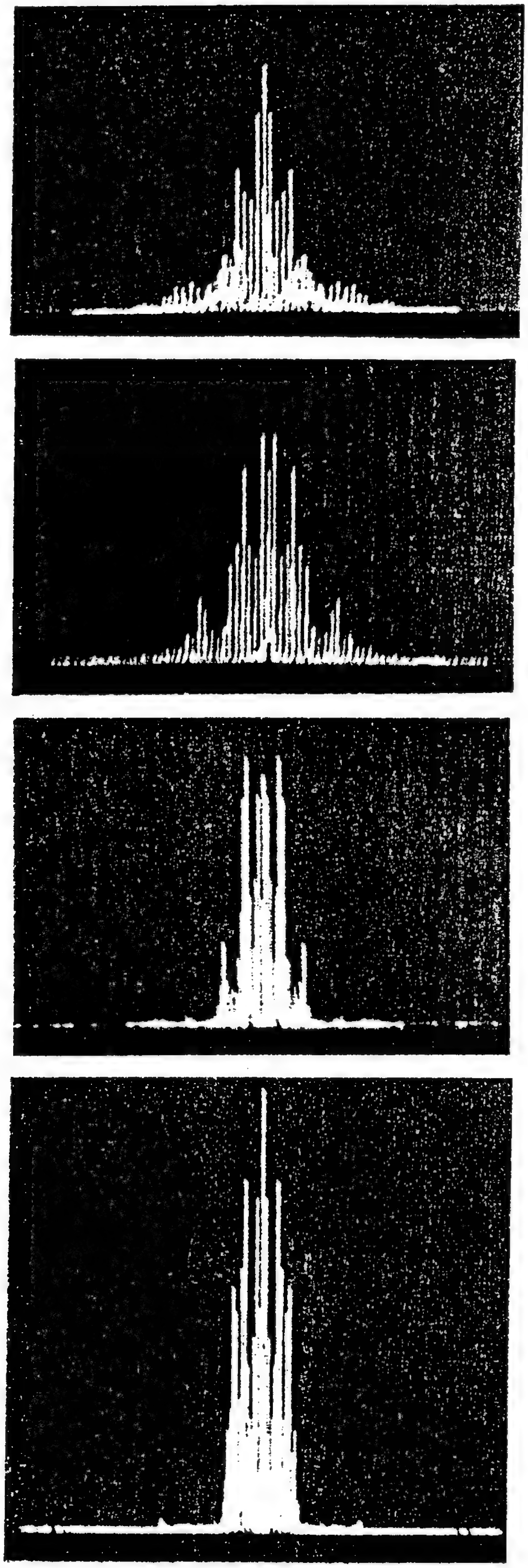


Figure 3: Frequency spectra of the monosounds *fi*, *re*, *bomb*, *ter*.

A simulated speech is generated in the form of a sequence of matrices representing the monosounds in the speech, in which the sounds *fi-re, fi-re-bomb, figh-ter*, and *bon-fi-re* are included at random locations, and with ten percent randomly damaged. The fuzzy pattern matching machine was activated and the detected keywords were recorded. This simulation was repeated 50 times and the results are recorded in the following table.

Keyword	correct detection	incorrect detection	non detection
fire	38	4	8
firebomb	32	5	13
fighter	36	3	11
bonfire	35	5	10

Thus, on average, the successful rate of the machine in detecting a predetermined keyword is around 70%, which is quite encouraging.

References

1. Aho, A. and Corasick, M. (1975). Efficient string matching: an aid to bibliographic search. *Communications of the ACM*, Vol. 18, No. 6, 333-340.

2. Furui, S. (1989). *Digital Speech Processing, Synthesis, and Recognition.* Marcel Dekker, New York, 54-55.

3. Le, T.V. (1993). *Techniques of Prolog programming.* John Wiley & Sons, New York, 199-203.

4. Le, T.V. (1993). Fuzzy pattern matching and its application to system simulation. In *Proceedings of the Fourth Annual Conf. on AI, Simulation, and Planning in High Autonomy Systems*, Sept. 1993, Tucson, Arizona. IEEE Computer Soc. Press, 54-58.

5. Markel, J.D. and Gray, Jr. A.H. (1976). *Linear Prediction of Speech.* Springer-Verlag, Berlin, 159-160.

6. Weiner, K. (1993). *Sound Effects Playhouse.* Waite Group Press, Corte Madera, CA.

Session 2H:

Test and Evaluation

Automatic Performance Monitoring and Evaluation

Peter Drewes
Statistica Inc
Orlando, Florida

Avelino Gonzalez, Ph.D.
University of Central Florida
Orlando, Florida

Abstract

With Distributed Interactive Simulation (DIS) there is an opportunity for large scale Computer Based Training (CBT). As more simulators interact through DIS, the complexity of the training systems increases. This, in turn, increases the work load on the instructor. Students are performing more complex tasks that cannot be viewed at a glance to determine their state. If an instructor has several students to monitor, it is difficult to track what each student is doing at any given time. One solution to this is to offload the instructor by interpreting and evaluating the student actions via computer programs. This can be accomplished through the use of advanced computer interpretation techniques. This interpretation will present to the instructor information concerning what and how the student is doing, without the need for the instructor to directly monitor the student.

Introduction

There are four major elements in the monitoring and evaluation of the CBT system: 1) How to automatically determine the optimal course of action to take under a particular set of circumstances (called the "goal standard"); 2) how to interpret what the student is doing; 3) how to compare what the student does compared to the goal standard, and; 4) how to control or give feedback to the students concerning their performance compared to the goal standards. Fortunately, most goal standards are defined and in place. Control and feedback to the student can be handled by the instructor; however, the difficulty lies in determining what the student is doing, and comparing it to what the student is supposed to be doing.

Rule-based systems appear to be a good means to interpret what the student action is, and how it relates to the goals of the CBT. But there are severe limitations concerning what a rule-based system can do. If a flight simulator is used as an example, there are many rules that would need to be incorporated to effectively monitor what a student is doing. An alternative to a rule-based system is a *template-based* approach. In this approach, which is the subject of this paper, templates represent small portions of the entire environment and can be directly related to some portion of the computer-based training goals. These templates are used to track student actions as they relate to the training goals. The student will progress through templates much as they would progress through scenarios in lessons. This direct student monitoring and evaluation can provide real-time feedback that is available for the instructor. By presenting the current template status, an instructor may view the student's progress and performance through the lessons.

Background

In the early days of computer modelling, the modelling process itself took most, if not all of the computer's resources. This left little capacity for carrying out auxiliary tasks. As the speed and efficiency of computers increased, the focus become how to optimize the training of the student. The instructor was left to monitor the student directly. This is especially true in flight simulation where the student-to-instructor ratio has almost always been one-to-one. The conventional means to convey information to the instructor about the student's progress through lessons or scenarios has been to present the instructor with what the student sees. This is done through visual repeaters, or instrument duplicators. There are also overview maps, or *God's Eye view* of simulations to assist the instructor in seeing the big picture. However, the instructor must still

0-8186-6440-1/94 $04.00 © 1994 IEEE

analyze what the student sees and compare it to the instructor's own knowledge. This can later be applied against what the training goals were. The largest drawback is that the instructor cannot monitor more than one student at a time. If there are several students in a classroom, most will go unmonitored. When an instructor changes from one student to another, he must transition from the original student's lesson and performance to the that of a new student. During this transition time, the instructor may not be able to monitor any students, and may become frustrated and ineffective if bombarded with help requests, or other distractions while trying to transition to the new student.

Problem Definition

The process of monitoring a student consists of four separate steps:

1) Determining the goals of the lesson or scenario that are being monitored - what the student should be doing.
2) Interpreting what the student is actually doing from his actions.
3) Comparing what the student is doing against the goals that were set out in step 1.
4) Providing feedback to the student. In other words, if the student has deviated from the goals, how can the training goals still be met.

The first step is to determine the goals of the lessons or scenarios. These goals are typically defined by the CBT. The objectives of the computer based training system become the high level goals of the system. For example, if the CBT was a flight trainer for student Instrument Flight Rules (IFR) practice, this would become the main goal. Subgoals that could be derived from the main goal is the desire to not stall the aircraft, i.e. slow down so much that the wings cannot support the weight of the aircraft. Even though it is possible to complete the Instrument Landing System (ILS) practice safely while stalling the aircraft, it is not desirable to do so. Subgoals should be kept to a minimum, and at such a level so that they do not hinder the original training goals.

The second step is to determine what a student is doing. This involves knowing what the student *should* be doing which relates back to the goals of the CBT. It is possible to look at all of the inputs that are provided to the student, and the output that the student gives. However it is difficult to analyze all of the information at once. There would be several solutions to this problem including duplication of the simulation code at another place to interpret what is happening, embedding the interpretation program within the simulator's program, or determining a way to interpret what the student should be doing from the normal output of the simulation.

The third step is to compare the student actions against the goal standards. This is done by comparing the interpreted student action from the second step with the goals of the CBT. Since the student actions are being analyzed from the standpoint of the training goals, it is possible to relate certain student functions to invalid steps, or goal violations. It is not desirable to compare each student action, as this could create an overly complex monitoring program. Moreover, from a training standpoint, certain things may not be important. By concentrating only on certain aspects of the training goals, and relating the student actions to these goals, a minimal monitoring program may be created that is still effective.

The fourth step is to solve the problem of what should be related back to the student; in other words, student feedback. If the student cannot perform the task properly, should the training environment be made easier? It is possible to create large tables or performance comparisons to determine what should be presented to the student. However, this becomes another difficult task. The instructor is a valuable part of the training environment, if the instructor can be provided with information from sections one through three, then this summary of events can make the transition from one student to another much easier. It allows the instructor to monitor the *monitoring* program to oversee what is going on within his or her classroom.

It is possible to attack this problem using an expert system. This system would create a set of rules or guidelines based on the actions of expert instructors. However, the decisions of an expert instructor are difficult to duplicate in a real-time environment. There are many rules and special cases that would have to be taken into consideration. This approach could yield an effective training environment, but not without a large effort in trying to duplicate instructor knowledge in the area.

Solution Outline

The first step in solving this problem is to determine the optimal course of action that the student should take under a particular set of circumstances. This area is basically covered through the creation of the CBT system. The training goals are defined here, at least on a high level. The training goals must be broken down into subgoals until they are independent of any other goal in the CBT. The idea is to have defined unique goals. The breakdown of the goals will not be discussed further, since it is not the focus of this paper.

One approach to the problems described in items two and three above is to represent the domain knowledge as a series of templates, each of which defines a task or procedure within the domain. These templates become a collection of generic information about student actions. These actions that may be combined to create a package that can describe what the student is specifically doing, based on generic events. An example in the automobile driving domain would be a template containing the actions: 1) Student pushes automobile clutch to floor, and 2) Student turns key in ignition switch. The first event does not specifically indicate that the automobile is to be started, the clutch is also used to change gears. The second event does not specifically indicate that the automobile is going to be started, it *may* be the student's desire to turn on the radio, which also involves the key in the ignition switch. By combining the two actions, there is a very high probability that if both actions take place, then the student is trying to start the car.

We will refer to a program which incorporates these templates as an *Artificial Intelligent Instructor* (AII). These templates can be used to guide the monitoring program through different segments of the training scenario, looking for things that cause either training violations, or transitions to another segment of the scenario. This is continued until the scenario is complete, or the student has left the training area of the CBT. The third step of the solution is comparing what the student is doing against the goals of the CBT. This is the first level breakdown of the training goals. The second step of interpreting what the student is doing. This involves breaking down the goals into events and will be discussed later.

The third step involves tracking the scenarios through the templates and providing the information to the instructor about what the student is doing. This in turn can be used to determine if the student is on or off course with respect to the training goals. If the student deviates from a training goal, he would leave an acceptable template, and move to a transitional template. This template would indicate a condition has been created that is in potential conflict with the training goals. An example of this in the flight instruction domain would be a student attempting to perform a pre-landing functionality check before reaching the destination airport. A condition exists that could result in training violations if it continues. This becomes a translation zone. The AII program would indicate to the instructor that the student is now in the translation zone, because of the task that may be incorrect for this phase of flight. If the student's altitude returns to an acceptable value, the AII would indicate this to the instructor, and the program would transition back to the acceptable template. Once this happens, the AII would stop reporting, since it is within the same template. If the student performed an unacceptable training task, the AII would determine that a condition has occurred, and will report the information to the instructor. It would be up to the instructor (or another program) to determine when the simulation needs to be stopped and the student informed of the situation.

The basis for the template guideline is determining what is important at what time. By determining different phases of the CBT lesson, it is possible to then break the overall training goals into sub-goals which support the training goals. These subgoals make up part of the template. What things should be of concern now. For example, if the IFR flight is used in the cruise phase of flight, only altitude and position are of relative concern. If the altitude or position deviates from the desired path should something be presented to the instructor. It does not matter that the student is making fuel/air mixture adjustments, or recalibrating the navigation instrumentation at this time. These events may be important for other phases of the training environment, but here they are not. All system output variables are effectively ignored, until a template that is concerned with the variables wakes up and analyzes them. Certain system constraints are kept within a separate group of templates that are checked each time. These include things like exceeding structural limits, crashing into the ground, or other system violations that do not depend on the phase of training. In this way each template may be kept generic, and possibly reused in

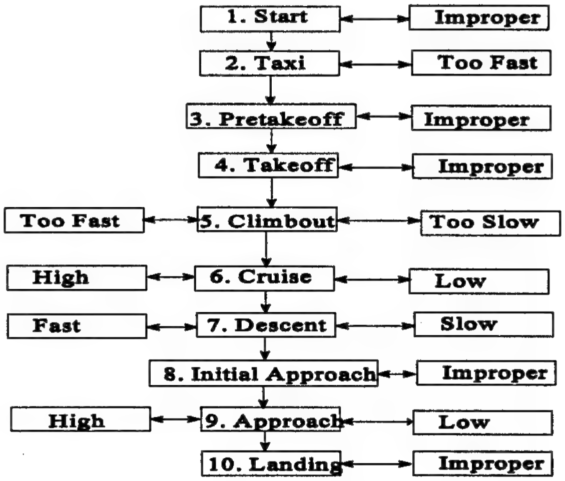


Figure 1 - Transition Template

another area of the monitoring. Each template would then have information concerning what happens when a parameter is invalid, or some other constraint exists that would cause the template to no longer be valid. The cruise template would not be valid as the student approaches the airport and begins to descend. The descent and/or proximity to the airport would trigger the template to release monitoring to the next template, which might be initial approach phase. This template would be concerned with reaching a particular approach point at a given altitude and velocity. Different system parameters would be evaluated against the sub-goals. Each of the subgoals relates back to the goal of the CBT, in this case, the practice of IFR flight with approach to landing.

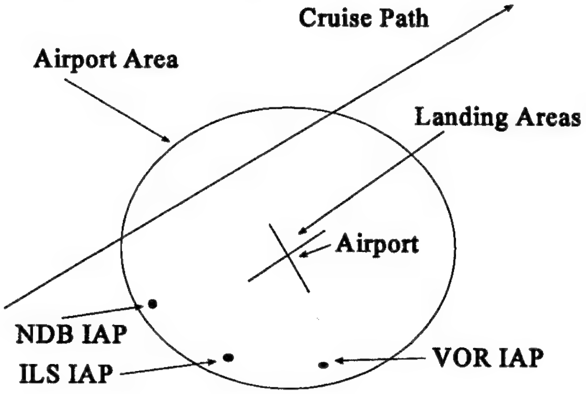


Figure 2 - Sample Airport

Through the examination of transitions between the subgoals a basic transition template can be created. This transition template is used to follow what is happening relative to the training goals. This is displayed in Figure 1. From the cruise template, it is possible to transition to the descent template, or invalid regions indicated by the altitude too low/high templates. For this case the cruise template would contain information about the simulation necessary to transition to the next template. Since the template is kept generic, information may not always be strict information. The cruise template may not have cruise values of 7000 feet minimum and 8000 feet maximum, but may have information passed from previous template, such as cruise value of Climbout_Template_Cruise - 500 feet minimum to Climbout_Template_Cruise + 500 feet. The Climbout_Template_Cruise is the altitude that was necessary to leave the climbout template and enter the cruise template. This tracking of transitions is done until the simulation is complete. By presenting the status of the current template, the instructor is able to determine what the student progress is within the training goals.

The second problem, interpreting what the student is doing, depends on the training goals, and is a breakdown on the third problem. The template list in Figure 1 describes different high level events that may take place in an instrument flight trainer. However, within those events, there needs to be a way to determine what the student is trying to do, and even predict what the student may be doing next. This is done to provide as must information about what is happening in the CBT without actually watching the student at all times. Using the instrument flight trainer example, there are several ways to approach an airport. The transitional point from cruise to approach to landing is called the initial approach point (IAP). For any given airport, there may be several IAPs. In Figure 1 the IAP is listed as a transition from descent to approach. However, since there may be several points to transition to, and several approaches that may be accomplished, the student actions and more importantly *intentions* must be determined. The diagram of the cruise path past a sample airport is listed in Figure 2. This describes an airport with three different types of landing facilities: Non Directional Beacon (NDB), Instrument Landing System (ILS) and Very-high Omni Range (VOR). These instruments can guide the pilot to the desired runway for landing. If the student is flying northeast (lower left to upper right) and begins to descend, it is possible that several things are happening: an NDB, ILS or VOR approach is about to take place, the student sees the airport and is going to land without instrument assistance, or the descent does not mean anything to this airport, the student is descending for another reason. If the student's path also changes as in Figure 3. Then it is possible that the

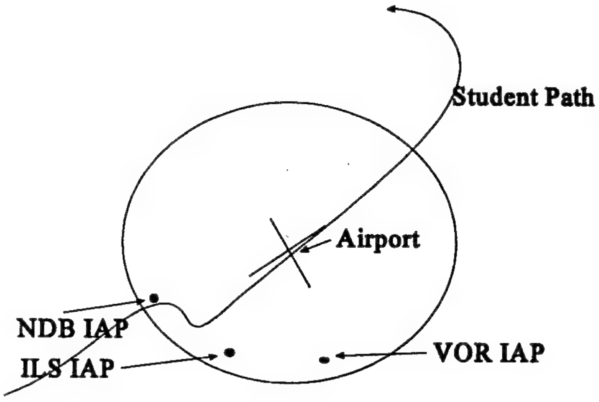


Figure 3 - Student Path

student is implementing the NDB approach, since the student passed the Initial Approach Point (IAP) of the NDB approach. This would be the indication as the student passes the NDB IAP. However, as the student's path is tracked, the student is not flying the pre-determined path for the NDB approach as illustrated in Figure 4. The path is similar to both the NDB and the ILS approach, and it does not appear that the student is attempting a VOR approach. However, the other two cases may still be possible. The student may be flying without the aid of the instruments, or may have changed course and altitude for reasons not dealing with the approach. Other parameters would be used to fill out each template. Since each of the above approaches uses different radio frequencies, these may be part of the templates. If the student is only tuned to the NDB frequency, that increases the probability that it is an NDB approach. However, if all of the frequencies are being monitored, all of the templates may continue to be filled in. Since there is not enough information here to determine what is happening, the template monitoring system must wait until something happens to make a decision on the events. As the student passes the airport at an altitude of 500 feet above the airport, a determination may be made at this point. Since the altitude is within the airspace of the airport, the general assumption is that the student must have been trying to land. With the radio frequencies tuned in, indicating a possible approach, a determination that the student was not implementing a VOR approach may be made because of his position. A further examination would require the instrument readings. If the ILS instruments are properly tuned for the approach, the student was executing an ILS approach, if both the NDB and ILS instruments are properly tuned, then the student was also probably executing an ILS approach, even thought the IAP point was missed, the approach was *closer* to the ILS than the NDB approach. This determination is made because the flight path is closer to the ILS than the NDB approaches (Figure 4). The rest of both the ILS and NDB templates are equal. Thus, the template that most resembles the student's observed actions will be deemed to represent his intentions. However, this conclusion is based on student actions, in the current environment. It is possible that the student wasn't using the instruments, and was only flying according to what he saw. However, the templates do provide a good measure of determining what the student is trying to do without interrupting the student's training to ask them.

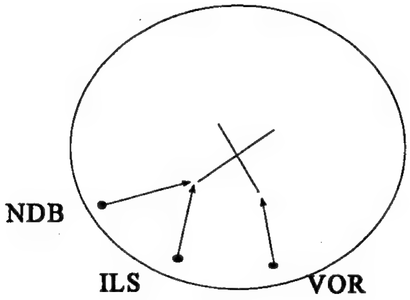


Figure 4 - Approach Paths

Using templates does not eliminate the need for a rule-based system, only diminishes it. The templates look at the rules from the training standpoint. From that perspective, important training parameters may be grouped. If the CBT does not care about certain functions or procedures that the simulation may perform, then it is not monitored. However, there must exist certain guideline parameters that would be enclosed in the *base templates.* These would be the general rules that the CBT must follow. For instance, If the student was descending because he was out of fuel, and the course and altitude deviations were due to this, we would not want to ignore these important conditions. However, these conditions may not be part of any of the training goals. The training goal may be landing practice, this may not include the need for fuel management. If outside parameters will be modelled, they would be included here. These parameters would be part of the base template structure, and would allow the other templates to be as generic as possible. The general processing of the information would include the gathering of information from the simulation, comparison with the generic templates, then comparison with the base templates. The base templates would be of the same nature as the generic templates, but they may not be the same for different trainers. The generic template would be good for any aircraft that contained NDB, ILS or VOR instruments. However, fuel starvation conditions and responses would be different for different aircraft.

The fourth problem is providing feedback to the student. This presents an interesting challenge. Since the second and third problems provide a good estimation on what the student is doing, a follow-on program could be used to present information back to the student concerning his or her progress and deviations. However, this is an estimate against the training goals. It would be better to provide this information to the instructor to interpret and in turn present to the student concerning the performance at the

CBT system. The information concerning student performance could be used to simplify the scenarios in real time if the student is struggling, or make it more difficult if the student is doing well.

System Integration

There are a large number of CBT environments for which the AII could provide productive assistance. However, the largest difficulty is integrating this new system into an existing trainer. There are two ways of integrating the system: at the simulator, or at the instructor station.

By integrating the AII at the simulator, it has access to everything that the simulation generates. This provides a thorough exposure to all variables within the simulation. There would be minimal time lags at the AII due to its close computational proximity to the information. It could actually be scheduled within the simulation time frame to provide as little impact as possible on the overall simulation. This approach suits well simulations that are changing rapidly, and the analysis of events needs more information than is currently being passed to the instructor. There is a larger time lag in presenting the information to the instructor, since the information must be passed from the simulation back to the instructor. However, the cpu load of the AII would be split between the simulator and the instructor station.

Integrating the AII at the instructor station provides a smaller system impact. If the simulation is real-time, it will not be affected by the AII running at the instructor station. However, the appropriate data must be transferred from the simulation to the instructor station already to make sure there is no system impact. For instance, if there is a digital repeater of the flight instruments in the instruments flight trainer, the information that is being passed to the instructor's station has all of the necessary information to determine the student's course of action. The AII would intercept the information and summarize it for the instructor. The cpu load of the AII would be taken completely by the instructor station, and not effect the simulation at all.

Summary

The template based monitoring systems allows the instructor to see what is going on within the classroom without the problem of examining every step that each student takes. A sample of the AII transition is shown below:

- Start
- Taxi
- Takeoff
- Climbout
- Cruise
- Descent
- Reached NDB IAP
- Course most like ILS approach
- Left Fuel Tank starvation

Since this simulation is time-based, each transition would have time stamp information. This transition would indicate that the student is potentially having problems with fuel management. This could be observed by looking at the fuel tank when the fuel ran out. However, by looking at the time stamp for the different transitions, an analysis of what phase took longer than was expected. The fuel starvation phase of flight can be made without the need for direct instructor intervention in the CBT. The instructor could be informed of the situation without viewing the student's instruments. This concept applies to not only Instrument Flight Practice, but to any area of simulation based training where the goals are defined. If it is applied to a maintenance trainer, the goal may be to measure the output frequency of a piece of equipment. This becomes one of the subgoals. There are many ways to perform maintenance, and it is not feasible to try and cover all of the possibilities. By subdividing the system goals into manageable sections, the AII can view the CBT from the goals standpoint, and provide a summary of student actions. It is not realistic to attempt to model all of the different ways that two aircraft can engage each other. However, if the goal is the engaging and destroying the enemy, this provides a more defined goal set.

As the number of non-generic rules increases, the base templates will increase to handle the special cases. As the base templates become larger than the generic templates, then the system has reverted back into a pure rule based system, and either the training goals must be re-evaluated, or another system will be needed. The template system does have the need to understand the training goals thoroughly before any system modeling can begin. Since the entire system is based on presenting the training goals to the instructor, as performed by the student, a complete requirements analysis must be done.

References

1. Levi, S., *Real Time System Design*, McGraw-Hill Publishing, 1990

2. Gonzalez, A. and Myler, H., *An Intelligent Instructor Support System for Training Simulators*, Proceedings of the Society of Computer Simulation Multi-Conference, March 1988

3. Thurman, R., *Applying Artificial Intelligence to Training Air Combat Maneuvering: The Potential, The Pitfalls, The Products*, Air Force Armstrong Laboratory. 1992

4. Smith, C. and Corripio, A., *Principles and Practice of Automatic Process Control*, Wiley Publishing, 1985

5. Rich, E. and Knight, K., *Artificial Intelligence*, McGraw-Hill Publishing, 1991.

6. Lin, S. and Dean, T., *Exploiting Locality in Temporal Reasoning*, Brown University/National Science Foundation, 1993

7. Riesbeck, C. and Schank, R., *Inside Case-Based Reasoning*, Lawrence Erlbaum Associates, 1989

8. Santos, E., *A Linear Constraint Satisfaction Approach to Cyclicity*, Brown University/Office of Naval Research, 1992,

FSATS: an Object-Based Approach to Distributed Interactive Simulation for C3I Test and Training

Eric Evans
Applied Research Laboratories
The University of Texas at Austin

Abstract

FSATS is a tool designed to support both testing and training for C3I systems. It provides the capabilities for collecting C3I tactical message traffic, reducing it for later evaluation, and interactively simulating C3I units. FSATS hardware consists of a variable number of LAN-based processors, and its software may be distributed among these processors in a range of possible configurations. The software is developed using an object-oriented model, where application-level functions are implemented as distinct object classes. FSATS interactively simulates C3I units by modelling each type as a set of logic tables and state data. Each such model is implemented by an object class. Object interaction is mapped onto a FSATS-provided message-delivery service. Objects invoke operations on each other by sending request messages, and the results may be returned in response messages. FSATS's object interaction model will probably migrate in the future toward compliance with industry standards, in order to take advantage of third-party software.

Background

The Fire Support Automated Test System (FSATS) is a distributed hardware and software system. It is used as a tool to support both testing and training for military Command, Control, Communications, and Intelligence (C3I) Systems. FSATS's sponsor is the Army Program Manager for Instrumentation, Targets, and Threat Simulators (PMITTS), and its intended users are the military fire support test and training community. FSATS was originally developed to support the testing and evaluation of the new Advanced Field Artillery Tactical Data System (AFATDS), an enhanced C3I system sponsored by the Army Program Manager for Field Artillery Tactical Data Systems (PMFATDS).

C3I systems can be complex and therefore costly to test. Reasonable test objectives for a C3I system might easily require battalion- or brigade-sized deployments of personnel and equipment to approximate the conditions in which the System Under Test (SUT) is to be used. Several different kinds of tactical communication networks may be involved, and there may be many instances of each. Further resources may be needed for monitoring and collecting data on the test itself. FSATS addresses this problem with its three main capabilities: (1) monitoring tactical communication networks and collecting the resulting tactical message traffic; (2) archiving the message traffic and reducing the collected data to a manageable form so that the SUT's performance can be evaluated; and (3) interactively simulating fire support units for the purpose of reducing the personnel and equipment resources needed to thoroughly exercise the SUT (thereby reducing the overall cost of the test). This paper focuses on FSATS's interactive simulation capability.

In addition to supporting the testing of C3I systems themselves, FSATS's simulation capability could also be used in a training role to "surround" a live player with simulated units. It could be configured to reinforce specific desired actions and procedures on the part of the live player. It could also help to fill out a battalion-, brigade-, or division-sized training laydown, providing the necessary realism at a reduced cost.

FSATS has been used by its customers to support several AFATDS tests in 1993 and 1994. Some of the lessons learned about FSATS during these tests are discussed throughout the paper. FSATS has not yet been employed in a training capacity.

ISBN 0-8186-6440-1. Copyright (C) 1994 IEEE. All rights reserved

0-8186-6440-1/94 $04.00 © 1994 IEEE

Architecture

There are three dominant aspects of FSATS architecture: it is distributed, it is highly configurable, and its software is object-oriented. Its distributed nature is apparent in both hardware and software. FSATS hardware generally consists of a collection of UNIX workstations[1] interconnected by a standard IEEE 802.3 local-area network (LAN). At runtime, FSATS software consists of separately executing programs that communicate primarily by exchanging messages. These programs may execute on the same host or on different hosts on the network. Currently FSATS uses standard transport facilities provided by the UNIX operating system for this message delivery[2]. Some information is also shared via a commercial database management system (DBMS) that is accessible across the network[3].

FSATS may be used in a wide range of hardware and software configurations. This flexibility is required so that it can accommodate the many possible SUT configurations as well as the different functions required for various test or training scenarios. A given FSATS hardware configuration may be as simple as a single workstation with an interface to a single SUT tactical network, or it may consist of many networked workstations attached to many tactical networks. FSATS software may be configured using its test planning feature to determine whether certain functions must be present and how they should be allocated to processors for a given test or training scenario. The resulting configuration is then constructed when FSATS execution begins.

FSATS software was designed using an object-oriented development model. This model was chosen early in the project because the advantages of object-oriented design (especially modularity and code reuse) were obviously useful in the development of a complex, large-scale software application like FSATS. The FSATS object model is essentially the same as the Core Object Model described in the Object Management Group's (OMG) Object Management Architecture (OMA) [2]. An FSATS *object* is a software entity that maintains its own state, and provides a well-defined set of *operations* that it will perform if requested to. A particular object is an instance of an *object class* and is specified by its unique *object identifier.* Objects of a given class provide the same set of operations, but each object maintains its own state. An operation is essentially a procedure which may require input parameters upon invocation, and may also provide output parameters upon completion. A client (usually an object itself) invokes a given operation on a particular object by specifying the object identifier, the operation, and any input parameters for that operation. The client may also receive output parameters in return if the operation provides them.

The FSATS object model is not identical to the OMA Core Object Model, however. The main difference is that FSATS provides no mechanism for inheritance. The FSATS object model was developed during the early design and prototype phase of the project in 1990, without knowledge of the OMG's work. However, both models were designed to achieve very similar goals and were influenced by many of the same developments in software research and industry during the same period.

The various capabilities of FSATS are decomposed into distinct functions, each of which is then implemented as its own object class. FSATS can create an instance of a function by creating an instance of its corresponding object class. This allows an FSATS configuration to contain only those functions needed for a given test or training scenario, and to allocate them to specific processors. There are object classes that implement functions such as collecting data from SUT networks, logging collected data to secondary storage media, acquiring position and global time data from Global Positioning System (GPS) transponders, and managing internal FSATS resources. Finally, there are object classes that implement the C3I simulation models used in FSATS's interactive simulation capability.

Interactive simulation

FSATS simulation is based upon a set of requirements known in the fire support community as Operational Facility (OPFAC) Logic, which has primarily been provided to ARL:UT as Government Furnished Information (GFI). OPFACs are the fire support units that together comprise the echelons responsible for initiating, directing, and implementing artillery fire on the battlefield. They are interconnected by any of several different kinds of tactical networks, and they interact by exchanging "tactical messages" among themselves. From a simulation perspective, any OPFAC can be modelled as an entity that maintains its own state, accepts input events (usually tactical messages), and produces output events (also usually tactical messages). FSATS simulates various types of OPFACs in the categories of sensor, command and control, and firing units. Real OPFACs and FSATS simulated OPFACs interact solely through the exchange of tactical messages.

Each type of simulated OPFAC model is implemented as a FSATS object class, which employs its own subset of OPFAC Logic to determine its behavior. A simulated

[1] SCO Open Desktop 2.0, on Intel x86-based workstations.

[2] The Internet User Datagram Protocol (UDP) [1], accessed via the UNIX Transport Layer Interface (TLI).

[3] Oracle 6.36 for SCO UNIX

OPFAC receives each input event as an operation request from another object. This request may convey a tactical message sent by another OPFAC, which may be real or simulated. It may also be an FSATS-internal request, such as a command to initialize or shut down sent by a resource manager object. Based upon the received request and its parameters, and upon its own current state, the simulated OPFAC uses its portion of OPFAC Logic to determine which actions to perform. These may include changing the model's state and/or making requests of other objects. Each such request may itself convey a tactical message that is sent to another real or simulated OPFAC.

Simulated OPFACs may be stimulated by tactical messages from two sources: (1) other OPFACs (real or simulated); and (2) the Time Ordered Event List (TOEL). A TOEL is a list of tactical messages that are to be delivered to simulated OPFACs at scheduled times. TOELs are in essence "scenario scripts" that permit some control over the inputs to a test or training scenario. An object class called the "TOEL server" reads the TOEL and dispatches the specified tactical message to the given OPFAC at the given time.

Some of the simulated OPFAC models are complex and may store much of their OPFAC logic tables and state data in the DBMS. The DBMS currently used in FSATS is accessible from across the network, but it has only a single, central server. This has led to contention among simulated OPFAC objects for access to the DBMS server, especially during the start-up phase of FSATS when most of these objects are being created and are trying to initialize their model state data. Therefore, some of the performance advantage of implementing the simulated OPFAC models as distributed objects has been negated by the fact that a critical resource they all use is not itself distributed. Future FSATS work will investigate the use of DBMS strategies such as data replication to alleviate this bottleneck.

The types and formats of the tactical messages and the FSATS control messages exchanged by simulated OPFACs do not currently comply with the standards being developed for Distributed Interactive Simulation (DIS) by the Workshops on Standards for the Interoperability of Defense Simulations [3]. This compliance was not specified as a requirement for the initial version of FSATS, but future work will include studying these standards for their applicability and possible benefits to FSATS and its users.

Object interaction

The software component of FSATS that supports object interaction is called the Distributed Object Environment (DOE). The DOE provides a message-passing service that objects use to request operations upon each other, as follows. A *message key* is defined for every possible type of DOE message, which may contain one particular data type. A given operation on an object may map to one or two DOE message types: one key for the message that contains the operation request, and another for the message that contains its response, if there is one. The data type conveyed by a DOE message therefore contains either the input or output parameters of an operation. An example of this mapping is expressed in Figure 1, for a hypothetical operation *op1*.

```
Operation declared as:

  void  op1 (  in      type1  arg1,
               in      type2  arg2,
                  out  type3  arg3  )

Corresponds to:

  Request message with:   Key = key1  Data = (arg1, arg2)
  Response message with:  Key = key2  Data = (arg3)
```

Figure 1

The DOE message-passing service provides object *location transparency* (clients need only know the identifier for a given object, not where that object resides) and both synchronous and asynchronous modes of operation. A client invokes an operation on an object by creating a DOE message which contains the object identifier, the correct message key for the request, and the "in" arguments for that operation. It then sends the request message using one of the DOE SEND primitives. One form of SEND blocks the client until the object's response message returns, which bears the "out" arguments of the requested operation. The other form of SEND just delivers the request without blocking the client. This is used for operations that have no "out" parameters.

Object implementations contain the subprograms, or *methods*, that actually perform the requested operations. Each object implementation also contains code that receives each operation request message (using one of the DOE RECEIVE primitives), extracts the message key, then invokes the correct method for the given key, passing it the "in" arguments from the request message. If there are "out" parameters for the operation, then the object will also build the response message containing those results and send it back to the requesting client.

During execution, FSATS objects reside within executable programs as shown in Figure 2. In addition to the object implementation software, each program links in the DOE procedures which objects use for exchanging DOE messages. When a client sends a message to another

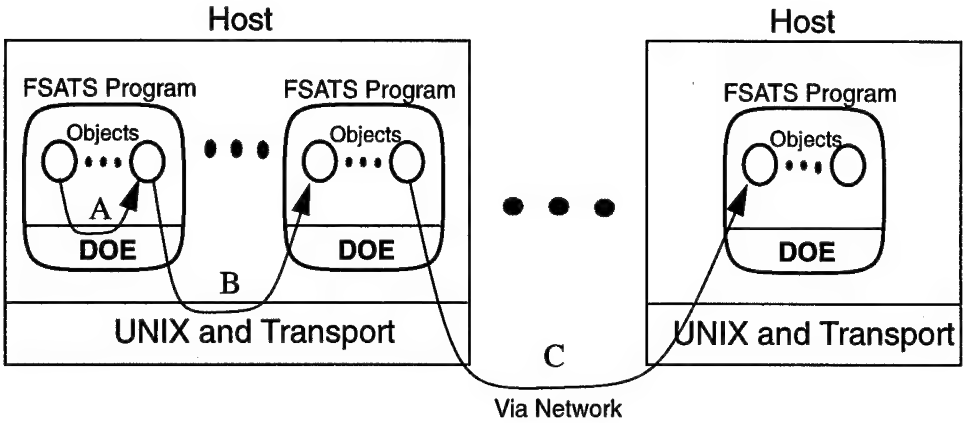


Figure 2

object, the client-side DOE code provides location transparency by looking up the destination object identifier in DOE's Object Directory (OD) service. This returns the location of the object's implementation. If the destination object is implemented in the same program, the request is delivered to it directly in memory (Figure 2, A). Otherwise the request message is relayed to the target object's server program via the UNIX-provided transport service (Figure 2, B and C). All messages sent via the transport are encoded using the standard External Data Representation (XDR) [4] so that they may be independent of different host data representations.

The DOE OD maps each object identifier in FSATS to that object's location (i.e., the transport address of its server program). Therefore, each object must register its identifier with the OD before any clients can send DOE messages to it. DOE's current OD implementation has no central server. The OD is replicated among all of the server programs. Each program contains an OD *agent.* A special protocol is used to ensure that all such agents comprising the OD quickly converge toward mutual consistency whenever an object changes its status, or whenever some inconsistency is detected. An object may change its status by registering or deregistering itself, in which case its local OD agent will notify all the others. If a client tries to send a message to an object identifier that is unknown to the client's local OD agent, that agent will broadcast a query for that object to all other OD agents. If the object in question has registered anywhere in the FSATS, its own OD agent will respond with the needed information. An OD agent will detect a routing error when a DOE message is delivered to an FSATS program which does not contain the destination object. It then sends a special message back to the source program notifying it of the error. The source OD agent will then use the query mechanism to resolve the error. OD agents can also detect attempts to register object identifiers which are already registered, in which case the source agent is notified.

Since the OD has no central server, there is no single point of failure in the system. Programs can be terminated during FSATS execution without disturbing anything but the objects that resided there and the objects that were interacting with them. We have found the OD protocol to be quite robust and nimble in dealing with OD changes and inconsistencies. However, this protocol relies heavily on the availability of a broadcast mechanism that can distribute a single message to all OD agents. This approach works very well in FSATS's current communications environment, since the Internet Protocol [5] provides a special broadcast address that can be used within a network, and the IEEE 802.3 LAN technology provides true broadcast support in hardware. Nevertheless, it would not scale well to FSATS configurations which included many hundreds of objects or more, and would be inefficient in a network that did not support true broadcasting.

In any case, DOE and its OD may be replaced in future versions of FSATS by third-party standards-based products. Like the FSATS object model with respect to the OMG's OMA, DOE was designed and implemented in the period 1990-1993, roughly the same time that OMG's Common Object Request Broker Architecture (CORBA) [6] was evolving. Because of the similarity in their design goals, DOE provides most of the services of an Object Request Broker (ORB) as described by CORBA, although it obviously does not comply with its specifications.

The ways in which DOE falls short of being a CORBA-style ORB are related to some persistent software development problems in FSATS. FSATS objects are currently not required to have explicit, well-defined interfaces. There is nothing like the Interface Definition Language (IDL) defined by CORBA. This means that a very useful feature provided by a CORBA ORB cannot be provided by DOE: automatic generation of the software that maps object operations to the underlying message-passing service. FSATS object implementors must write the so-called *skeleton* software that receives each DOE message, interprets its contents, invokes the correct method, and builds and sends the reply message. Client implementors must write each *stub* procedure that builds and sends the request message for a given operation, waits for the response, and returns the results. But given an explicit, well-defined object interface, both the client stub and object skeleton source code could be generated automatically. Client implementors would then need only use a procedure-call interface to invoke an operation on an object. Object implementors would only have to provide the actual methods for each operation. CORBA defines a tool called the IDL compiler, which takes an object interface defined in IDL and generates the client stubs and object skeleton. This source code can then be compiled and linked in to the client and object implementations respectively.

Automatic generation of the stub and skeleton source code would provide two important benefits for the development of the distributed object software in FSATS. For one, programmers would no longer need to write stub and skeleton source code by hand, which is a rather mechanical and error-prone process anyway. For another, it would provide a mechanism which enforces consistency between the operations that an object actually provides and the operations a client expects that object to provide. FSATS client programmers currently must rely on non-source code specifications, usually provided in comments. With an IDL compiler, both client and object implementation would start with the same interface definition for the object, so changes to the object's interface would be forced upon the client when the stub procedures are regenerated and re-linked.

Our realization of DOE's shortcomings compared to the emerging model for distributed objects has led us to develop an approach for moving FSATS towards industry standards in this area. We have chosen the CORBA standards as our target, because of the similarity to FSATS's object model, its industry support, and its source language bindings: C and C++ (for reasons outside the scope of this paper, future FSATS objects are likely to be developed in C++ rather than the currently used Ada).

The ultimate goal is to make future FSATS objects source-code-portable to any host/operating system platform for which there is an ORB implementation with the desired language binding. The main benefit of achieving this would be to provide FSATS the option of replacing our home-grown DOE with a commercially available and supported ORB. This would include replacing DOE's OD with one based on the Naming Service Specification defined in the OMG's Common Object Services Specification [7]. It would also allow FSATS to execute on a greater range of potential host and operating system platforms.

An important *interim* goal is to allow a migration period during which the old FSATS objects can still interact with newly-developed FSATS objects that expect a CORBA-compliant ORB. During this period, the new FSATS objects would actually use a CORBA-compliant version of DOE, which could still interoperate with the old DOE using the old DOE message-passing protocols. However, once all of the old FSATS objects have been discarded or ported to the new CORBA-compliant DOE, that DOE itself could be replaced with any third-party CORBA ORB.

Conclusions

In general, we have found that a distributed object architecture works well for supporting the distributed interactive simulation capabilities required of FSATS. Notwithstanding minor bottlenecks like the DBMS, performance has been satisfactory in the simulations to date. The flexibility of configuration inherent in the architecture has allowed FSATS to support different simulation scenarios, and has enabled the tuning of simulation performance by balancing the load of model objects across host processors.

The DOE system software that supports FSATS's distributed objects has worked very well for all of FSATS's functions, including distributed interactive simulation. Object interaction via DOE is fast and reliable enough that the overhead of object distribution has not been a problem. The distributed object directory is robust and adapts to changes quickly. However, experience during the last two years has highlighted areas which should be improved. The current object directory design would probably decline noticeably in efficiency as the size and complexity of the FSATS configuration grew. Also, DOE is lacking somewhat in the support of object software development and management. Therefore, future FSATS work will likely focus on migrating FSATS objects to work over any CORBA-compliant ORB, ultimately allowing FSATS to replace its proprietary DOE with whatever third-party ORB works on a required host and operating system platform.

References

[1] Postel, J.B. "User Datagram Protocol" (Internet RFC 768), 1980.

[2] Object Management Group. "Object Management Architecture Guide" (OMG 92.11.1), Object Management Group, Framingham, MA, 1992.

[3] UCF Institute for Simulation and Training. Distributed Interactive Simulation Standards (series), UCF Institute for Simulation and Training, Orlando, FL.

[4] Sun Microsystems, Inc. "XDR: External Data Representation Standard" (Internet RFC 1014), 1987.

[5] Postel, J.B. "DoD Standard Internet Protocol" (Internet RFC 760), 1980.

[6] Object Management Group. "The Common Object Request Broker: Architecture and Specification" (OMG 91.12.1), Object Management Group, Framingham, MA, 1991.

[7] Object Management Group. "Common Object Services Specification - Volume 1" (OMG 94.1.1), Object Management Group, Framingham, MA, 1994.

Author Index

Notes

Offices of the IEEE Computer Society

Headquarters Office
1730 Massachusetts Avenue, N.W.
Washington, DC 20036-1903
Phone: (202) 371-0101 — Fax: (202) 728-9614

Publications Office
P.O. Box 3014
10662 Los Vaqueros Circle
Los Alamitos, CA 90720-1264
Membership and General Information: (714) 821-8380
Publication Orders: (800) 272-6657 — Fax: (714) 821-4010

European Office
13, avenue de l'Aquilon
B-1200 Brussels, BELGIUM
Phone: 32-2-770-21-98 — Fax: 32-2-770-85-05

Asian Office
Ooshima Building
2-19-1 Minami-Aoyama, Minato-ku
Tokyo 107, JAPAN
Phone: 81-3-408-3118 — Fax: 81-3-408-3553

IEEE Computer Society

IEEE Computer Society Press Publications

Monographs: A monograph is an authored book consisting of 100-percent original material.
Tutorials: A tutorial is a collection of original materials prepared by the editors and reprints of the best articles published in a subject area. Tutorials must contain at least five percent of original material (although we recommend 15 to 20 percent of original material).
Reprint collections: A reprint collection contains reprints (divided into sections) with a preface, table of contents, and section introductions discussing the reprints and why they were selected. Collections contain less than five percent of original material.
Technology series: Each technology series is a brief reprint collection — approximately 126-136 pages and containing 12 to 13 papers, each paper focusing on a subset of a specific discipline, such as networks, architecture, software, or robotics.
Submission of proposals: For guidelines on preparing CS Press books, write the Managing Editor, IEEE Computer Society Press, P.O. Box 3014, 10662 Los Vaqueros Circle, Los Alamitos, CA 90720-1264, or telephone (714) 821-8380.

Purpose

The IEEE Computer Society advances the theory and practice of computer science and engineering, promotes the exchange of technical information among 100,000 members worldwide, and provides a wide range of services to members and nonmembers.

Membership

All members receive the acclaimed monthly magazine *Computer*, discounts, and opportunities to serve (all activities are led by volunteer members). Membership is open to all IEEE members, affiliate society members, and others seriously interested in the computer field.

Publications and Activities

***Computer* magazine:** An authoritative, easy-to-read magazine containing tutorials and in-depth articles on topics across the computer field, plus news, conference reports, book reviews, calendars, calls for papers, interviews, and new products.

Periodicals: The society publishes six magazines and five research transactions. For more details, refer to our membership application or request information as noted above.

Conference proceedings, tutorial texts, and standards documents: The IEEE Computer Society Press publishes more than 100 titles every year.

Standards working groups: Over 100 of these groups produce IEEE standards used throughout the industrial world.

Technical committees: Over 30 TCs publish newsletters, provide interaction with peers in specialty areas, and directly influence standards, conferences, and education.

Conferences/Education: The society holds about 100 conferences each year and sponsors many educational activities, including computing science accreditation.

Chapters: Regular and student chapters worldwide provide the opportunity to interact with colleagues, hear technical experts, and serve the local professional community.